UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Escola d'Enginyeria de Barcelona Est

TREBALL FI DE GRAU

Grau en Enginyeria Electrónica industrial i automàtica

DISSENY ELECTRÒNIC I POSTA EN MARXA D'UNA IMPRESSORA 3D AMB RASPBERRY

Memòria i Annexos

Autor: Meritxell Rodríguez Torrejón
Director: Jose Antonio Travieso Rodríguez
Co-Director: Sebastian Tornil Sin

Convocatòria: Gener 2018

Resum

En aquest treball de final de grau es realitzarà tota la part electrònica i de programació necessària per poder utilitzar una Raspberry Pi per controlar per complert una impressora 3d, i així poder obtenir connexió sense fils amb qualsevol dispositiu amb connexió a internet.

La Raspberry Pi és un dispositiu electrònic amb connexió a internet capaç de realitzar tasques complexes com un ordinador. Col·loquialment es diu que és el segon ordinador més venut del món. Ja que es pot considerar un ordinador.

Per tal de realitzar aquesta tasca, s'ha dut a terme el disseny electrònic, que comporta el software i el hardware totalment adaptat a les necessitats de la Raspberry Pi juntament amb Octoprint, que és el software que es fa servir per a dur a terme la comunicació via internet, i per poder controlar i observar el procés de impressió.

Per això s'han utilitzat com a referència els elements més comuns que es fan servir avui en dia en un sistema de impressores 3d basat en Arduino, que són software de l'Arduino, que s'anomena Marlin i la placa de potència que s'anomena Ramps.

En aquest treball s'explica que és una impressora 3d i els conceptes bàsics per entendre en què ens hem basat per dissenyar el nostre sistema electrònic.

En el cos del treball es podrà veure el procés detallat, les decisions i els inconvenients que s'han trobat per realitzar-lo.

Per concloure, es podrà veure el disseny final del Hardware, que l'hem anomenat Raspra. Y el resultat obtingut que en aquest cas ha sigut positiu.

Resumen

En este trabajo de final de grado se realizará toda la parte electrónica y de programación necesaria para poder utilizar una Raspberry Pi para controlar por completo una impresora 3d, y así poder obtener conexión inalámbrica con cualquier dispositivo con conexión a internet.

La Raspberry Pi es un dispositivo electrónico con conexión a internet capaz de realizar tareas complejas como un ordenador. Coloquialmente se dice que es el segundo ordenador más vendido del mundo. Ya que se puede considerar un ordenador.

Para realizar esta tarea, se ha llevado a cabo el diseño electrónico, que conlleva el software y el hardware totalmente adaptado a las necesidades de la Raspberry Pi junto con Octoprint, que es el software que se utiliza para llevar a cabo la comunicación vía internet, y para poder controlar y observar el proceso de impresión.

Para ello se han utilizado como referencia los elementos más comunes que se utilizan hoy en día en un sistema de impresoras 3d basado en Arduino, que son software del Arduino, que se llama Marlin y la placa de potencia que se llama Ramps.

En este trabajo se explica que es una impresora 3d y los conceptos básicos para entender en que nos hemos basado para diseñar nuestro sistema electrónico.

En el cuerpo del trabajo se podrá ver el proceso detallado, las decisiones y los inconvenientes que se han encontrado para realizarlo.

Para concluir, se podrá ver el diseño final del Hardware, que la hemos llamado Raspra. Y el resultado obtenido en este caso ha sido positivo.

Abstract

In this end-of-degree project, all the electronic and programming part necessary to use a Raspberry Pi to fully control a 3d printer will be carried out, so that you can get wireless connection to any device with an internet connection.

In order to carry out this task, electronic design has been carried out, which involves software and hardware that is fully adapted to the needs of Raspberry Pi along with Octoprint, which is the software used to carry out Internet communication, and to be able to control and observe the printing process.

For this reason, the most common elements that are used today in a system of 3d printer based on Arduino, which are Arduino software, which is called Marlin and the power plate called Ramps, have been used as a reference.

This paper explains that it is a 3D printer and the basic concepts to understand what we have based on designing our electronic system.

In the body of the work you will be able to see the detailed process, the decisions and the inconveniences that have been found to carry it out.

To conclude, you can see the final design of Hardware, which we have called Raspra. And the result obtained in this case has been positive.

Agraïments

Vull agrair aquest treball als companys de feina de Boloberry per donar-me l'empenta necessària per realitzar aquest treball.

També agrair el suport del Oriol que m'ha ajudat a enfocar-lo.

De igual forma que vull agrair la realització d'aquest treball al meu tutor Sebastian Tornil, perquè gracies a ell he tingut l'empenta final per poder acabar el treball.

I com no, agrair al meu suport constant Gerard per donar-me recolzament en tots els moments que ho he necessitat.

Índex

Índex de figures

1. Prefaci

1.1. Origen del treball

L'origen del treball es basa en la necessitat de realitzar un enllaç sense fils entre la impressora 3D i un dispositiu electrònic fixe o mòbil sense la necessitat de introduir un tercer element.

Avui en dia, per a poder realitzar aquesta connexió es precisa d'un element extern, ja sigui un ordinador o una raspberry pi connectada en el actual sistema electrònic de les impressores 3D ja que aquest per si sol no pot establir connexió amb internet.

1.2. Motivació

La motivació d'aquest treball sorgeix de la necessitat de poder tenir una connexió sense fils amb la impressora 3D d'una forma senzilla i pràctica per l'usuari.

L'elecció d'aquesta temàtica es deu a que a l'hora de realitzar les pràctiques en una empresa, aquesta es dedicava al disseny i fabricació d'impressores 3D, i va sorgir la necessitat de crear una unió entre la impressora i l'ordinador de una forma que fos més eficient de la que existia en aquell moment.

Per tal de solucionar aquesta necessitat s'ha enfocat al treball per eliminar el ús de l'actual sistema de comunicació amb la impressora 3D i crear un nou sistema a partir de una raspberry pi, per així poder realitzar la connexió sense necessitar de tenir connectada la impressora a un ordinador.

2. Introducció

2.1. Objectius del treball

L'objectiu d'aquest treball és realitzar un nou sistema electrònic, basat en el actual, però substituint el arduino mega, que és la part principal, per una raspberry pi, ja que amb aquesta es pot realitzar connexió a internet.

L'arduino mega és un dispositiu electrònic capaç de realitzar tasques complexes amb entrades i sortides analògiques i digitals.

Com a primers objectius específics es tractarà de implementar una etapa de potència dissenyada exclusivament per la raspberry pi, ja que actualment es exclusiva per l'arduino mega.

Com a segon objectiu específic s'implementarà un software capaç de realitzar totes les tasques que es realitzen amb la unió de l'arduino mega i la raspberry pi en un sol dispositiu, en aquest cas la raspberry pi, ja que actualment es precisen els dos elements units cadascú amb un software independent.

Com a tercer objectiu es realitzarà una interfície, basada en la actual amb la qual el usuari pugui interactuar amb la impressora amb una pantalla ja sigui connectada a la pròpia raspberry pi o la que ens proporcioni un dispositiu mòbil o fixe com pot ser un ordinador.

2.2. Abast del treball

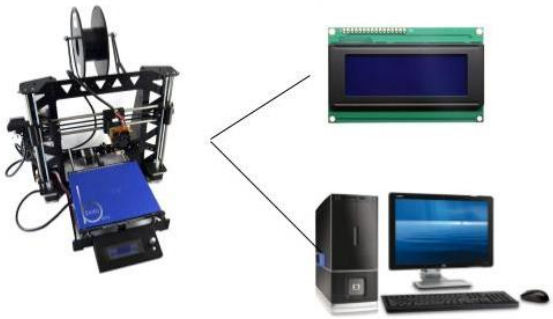
L'abast del treball és crear un nou sistema electrònic amb el qual poder controlar la impressora 3D, en aquest cas un model Prusa P3 20x20 amb llit calent i un sòl extrusor.

La connexió amb la impressora es podrà realitzar a través de la pantalla que es connectarà a la raspberry, o a través de qualsevol dispositiu amb connexió a internet.

La evolució de la comunicació amb la impressora amb els diferents dispositiu a tinguts diferents fases. Al principi la impressora només es podia comunicar amb l'ordinador:

Imatge 1: Primera fase de la comunicació amb la impressora

En la segona fase s'inclouria l'ús de una pantalla lcd o externa connectada a la impressora:

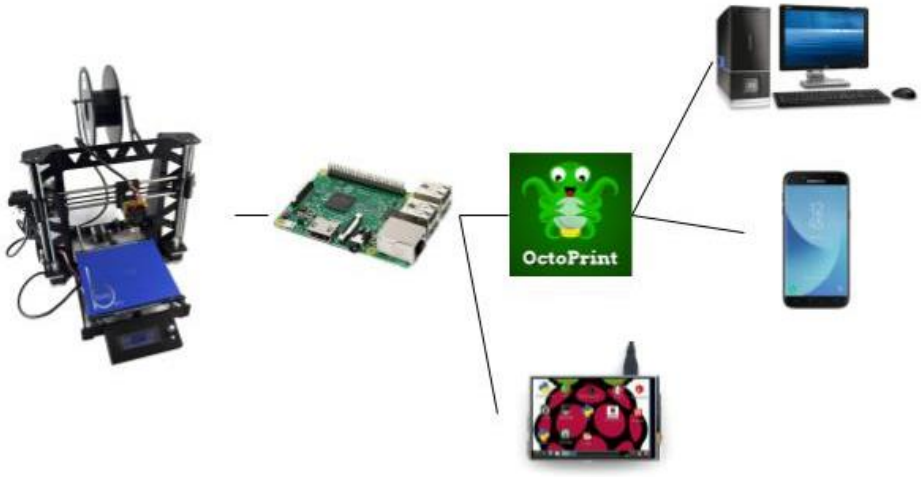


Imatge 2: Segona fase de la evolució de la comunicació

En la tercera fase s'incorpora l'ús de l'arduino que ha estat present en les altres dos fases:

Imatge 3: Tercera fase de la evolució de la comunicació

Finalment es mostrarà l'estat amb el qual es vol arribar amb aquest projecte:



Imatge 4: Fase final a la qual es vol arribar

2.3. Descripció del treball realitzat

El treball que s'ha realitzat al llarg de tot el procés, amb el propòsit de concloure satisfactòriament amb els objectius , en orde cronològic son:

- Recerca del disseny actual de la etapa de potència connectada a l'arduino mega en format digital.
- Adaptació a les característiques de la raspberry pi.
- Realització del prototip en format PCB.
- Aprenentatge del llenguatge python .
- Creació del programa basat en el marlin .
- Acoblament i instal·lació
- Avaluació de resultats.

Per realitzar el treball es requereix de coneixements en electrònica, programació i disseny de circuits impresos.

3. Impressió 3D

Una impressora 3D és un dispositiu capaç de realitzar un objecte en 3D a partir de un disseny 3D.

3.1. Descripció del funcionament

El funcionament bàsic d'aquests dispositius es a partir de la superposició de capes. L'element prèviament dissenyat en 3D, amb l'ajut de un programa, es transforma en un element creat a capes de un gruix especificat. La impressora materialitza les capes i les va sobre ponent.

Les impressores 3D es basen en el moviment opensource on el coneixement es de tots. Per tant les noves millores que es van trobant són penjades a internet per tal de que els usuaris de aquest tipus de tecnologia les puguin aprofitar en les seves impressores, i així entre tots poder fer evolucionar aquesta tecnologia que encara esta molt nova i aconseguir que sigui a l'abast de tothom.

3.2. Descripció dels tipus

Hi ha diferents tipus de impressores 3D segons la necessitat que es vulgui realitzar, com per exemple els FFF, SLA i SLS.

El model més comú es el anomenat de tecnologia FFF, aquest es basa en unes bobines de filament termofusible, amb el qual es fa passar per una superfície amb una alta temperatura capaç de desfer el plàstic i aconseguir que al tocar una superfície llisa es quedi solidificat, i d'aquesta forma crear el model 3D. Aquests model és el més comú, ja que és el més econòmic i permet la fabricació de la impressora a casa. Aquests tipus de tecnologia produeix les peces en un temps mitjà en comparació amb les altres i es poden aconseguir precisions de 100 micres en el eix Z i de 50 a 100 micres en el eix X-Y. Amb aquest tipus també es pot obtenir una gran varietat de materials, com per exemple el ABS, el PLA, el flexible, amb partícules de fusta, de fibra de carboni, de ceràmica, el Nylon, el PETG i diferents tipus que van sorgint cada dia a causa de la investigació. Un inconvenient de aquest tipo de tecnologia es que s'ha de tenir molt en compte la geometria del objecte en 3d que es vol imprimir, ja que si esta format per arcs o voladissos no es poden realitzar, ja que el plàstic no es pot quedar sòlid en el aire. Una solució es la implementació de suports que després es poden extreure, el problema es que es malbarati material i es poden quedar marques a la superfície que han estat enganxats.

El segon model es el SLA, aquest es basa en una tecnologia anomenada també com Estereolitogrfia, aquesta tecnologia crea el model en 3D a partir de un procés on un laser il·lumina un foto-polímer

líquid just en els punts on vols que es solidifiqui, i així capa a capa. Cada capa es diferent i es va solidificant a sobre de una placa immersa en bany de fotopolímer. Aquesta tecnologia també permet la utilització de diferents tipus de materials i amb bones propietats mecàniques. Aquest tipo de tecnologia té un inconvenient ja que es malbarati un 30% del material utilitzat, i un cop realitzada la peça a de realitzar un post tractament a sota d'una llum ultravioleta, ja que directament quan surt de la impressora la resina no esta solidificada al 100%. En aquest tipus de tecnologia no hi ha inconvenient en reproduir objectes amb arcs i voladissos, ja que el fluid foto-polímer flueix per el model solidificat i la impressió es realitza en sentit invers. A més, cal destacar que es un procés més lent que les FFF però s'aconsegueixen objectes amb una precisió de 60 micres en tots els eixos.

Imatge 5: Peça fabricada amb SLA

L'últim model anomenat, el SLS o sinterització selectiva per làser, aquest model també utilitza un làser, però en aquests cas es un model mes potent que sintetitza el pols en plàstic. El material més comú utilitzat es el Poliamida/Nylon, aquestes peces ofereixen una major resistència que les anomenades anteriorment. Amb aquest tipus de tecnologia no es produeixen pèrdues de material, ja que un cop s'ha realitzar la peça, el pols sobrant es torna a emmagatzemar per un proper us. Aquesta tecnologia també necessita un post tractament de la peça un cop finalitzada per la impressora.

Imatge 6: Peça fabricada en SLS

3.3. Descripció dels models

En aquest treball ens basarem en les impressores 3D de tecnologia FFF, dins d'aquest tipus hi ha una gran varietat de models en funció de tots els aspectes de la impressora que són modificables per obtenir millores i diferents condicions.

Per començar mencionarem els model que varien en funció de la estructura, podem observar tres models molt diferenciat, el cartesianes, el delta y el core xy.

El model més popular és el model cartesianes on el moviment de cada eix x, y o z es creat per un motor individual que és accionat des de la electrònica per un driver diferents. Estan dotades de una superfície de impressió quadrada que normalment s'encarrega de realitzar el moviment de un dels eixos. Aquest model de impressora és el més comú ja que són fàcils de utilitzar i hi ha molta informació en internet respecte errors, millores i guies de utilització. El problema respecte les altres es que la velocitat es inferior, la precisió es molt inferior, no es poden imprimir qualsevol tipus de disseny, sobretot els que estiguin formats per arcs o voladissos.

Imatge 7: Impressora 3d cartesiana

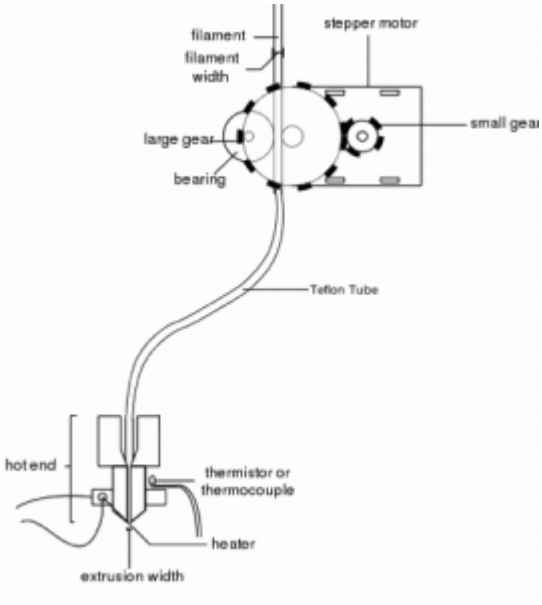
El segon model més popular és el delta amb qual el moviment de el capçal es realitza amb la unió de el moviment dels motors que són moguts pels divers dels eixos x, y i z. Els eixos estan posicionats de forma vertical i disposades de forma triangular. La impressió es realitza sobre una superfície rodona que es manté fixa. Els límits de la impressió estan definits per el diàmetre de la base y de l'alçada dels eixos. Aquest model de impressora té l'avantatge de que tenen una alta velocitat i alta acceleració, alta

definició, es pot redimensionar. El problema respecte els altres models és que no són fàcils de utilitzar, a causa de que la calibratge no sol ser perfecte i no hi ha gaire informació a internet.

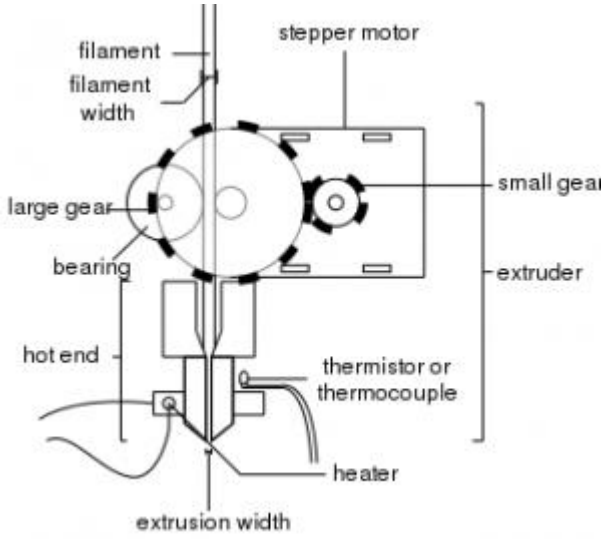
Y per últim el model corexy, en aquest model podem observar una barreja de els dos models anteriors, ja que el eix z es mou individualment i el moviment de el capçal en el eix x i y es mou amb la unió del moviment dels motors que són accionats pels divers x i y. En aquests model la superfície de impressió es quadrada i es la encarregada de realitzar el moviment en z. Aquest model presenta grans velocitats i acceleracions i presenta gran precisió.

3.4. Descripció de capçals

A part de aquesta classificació segons el mecanisme de moviment, podem trobar altres característiques en funció del capçal. Hi ha dos diferenciés bàsiques que es basen en el motor que fa empènyer el filament de plàstic cap al Hotend, que és on es troba el bloc tèrmic a alta temperatura. Si el motor està situat fora de el capçal en moviment rep el nom de bowden, sinó rep el nom de extrusió directe.



Imatge 9: Extrusor bowden



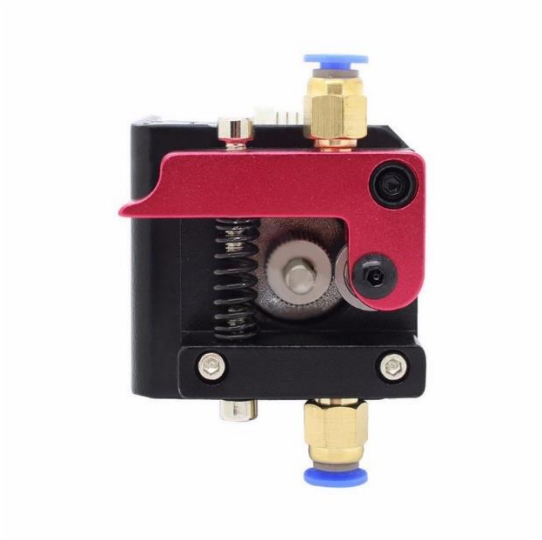
Imatge 8: Extrusor amb engranatge

En un model de extrusió directe es poden trobar molts tipus de extrusor diferent segons els mecanismes que s'utilitzin per unir el moviment del motor amb un engranatge per moure el filament.

Els més coneguts són els extrusors MK8, E3D i Jonas.

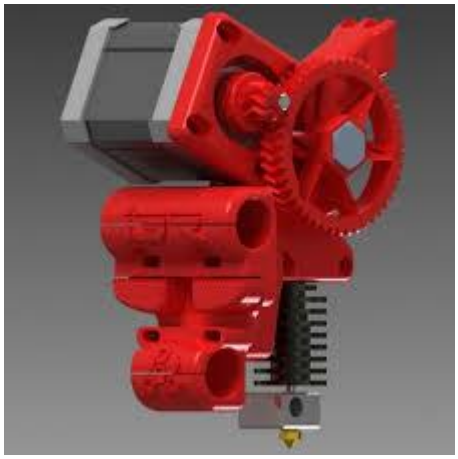
El extrusor MK8 consisteix en la unió de unes peces de metall on el engranatge que oprimeix el filament de plàstic per empènyer cap al hotend es troba just al eix principal del motor. Les altres peces permeten

regular lo molt que s'oprimeix el filament. Aquest model produeix gran precisió ja que el moviment del motor es directament aplicat sobre el filament. També produeix molt bon resultat per imprimir filament flexible ja que des de que s'empeny fins que arriba al hotend hi ha molt poc recorregut, per lo que s'aconsegueix que no tingui espai per doblegar-se.



Imatge 10: Extrusor MK8

El Jonas és un model indirecte on el moviment del eix del motor es reduït per una sèrie d'engranatges de diferents mides. Aquest tipus de extrusor es pot fabricar directament amb una altre impressora 3d ja que es realitza a partir de peces impreses.



Imatge 11: Extrusor Jonas

3.5. Descripció de propietats personalitzables

A part de les ja esmentades anteriorment es podem trobar una gran varietat de impressores segons els materials utilitzats per la seva creació (metall, fusta, metacrilat), la quantitat de extrusors, poden haver 1, 2 o fins i tot més si s'utilitzen dispositius de pàrquing.

Al formar part del mon de la tecnologia open source cadascú pot proporcionar les seves millores a la comunitat i així anar personalitzant la seva impressora segons les necessitats que es tinguin. Es pot personalitzar gairebé qualsevol part de la impressora, hem de tenir en compte que a partir de una impressora es poden imprimir noves peces per ella mateixa i així anar-la personalitzant al nostre gust. La forma de la estructura també és un element amb el qual es pot jugar molt, ja que dintre de les cartesianes hi ha diferents tipus de estructures que són totalment vàlides. També es pot personalitzar la part electrònica, ja que es poden canviar components, com els sensor de final de carrera, poden ser mecànics, òptics, inductius, es poden utilitzar per afegir un calibratge automàtic del eix z per mantenir la distancia entre el llit i el filtre del extrusor de forma automàtica. Aquest sistema es pot realitzar de moltes formes possibles i totes són vàlides, al final es l'usuari qui decideix quin model utilitzar. La mida de la base de impressió també es un element personalitzables on l'usuari pot decidir quina mida desitja utilitzar, la més comú es la de 20x20x20cm, tot i que avui en dia s'està tractant molt amb les 20x30x20.

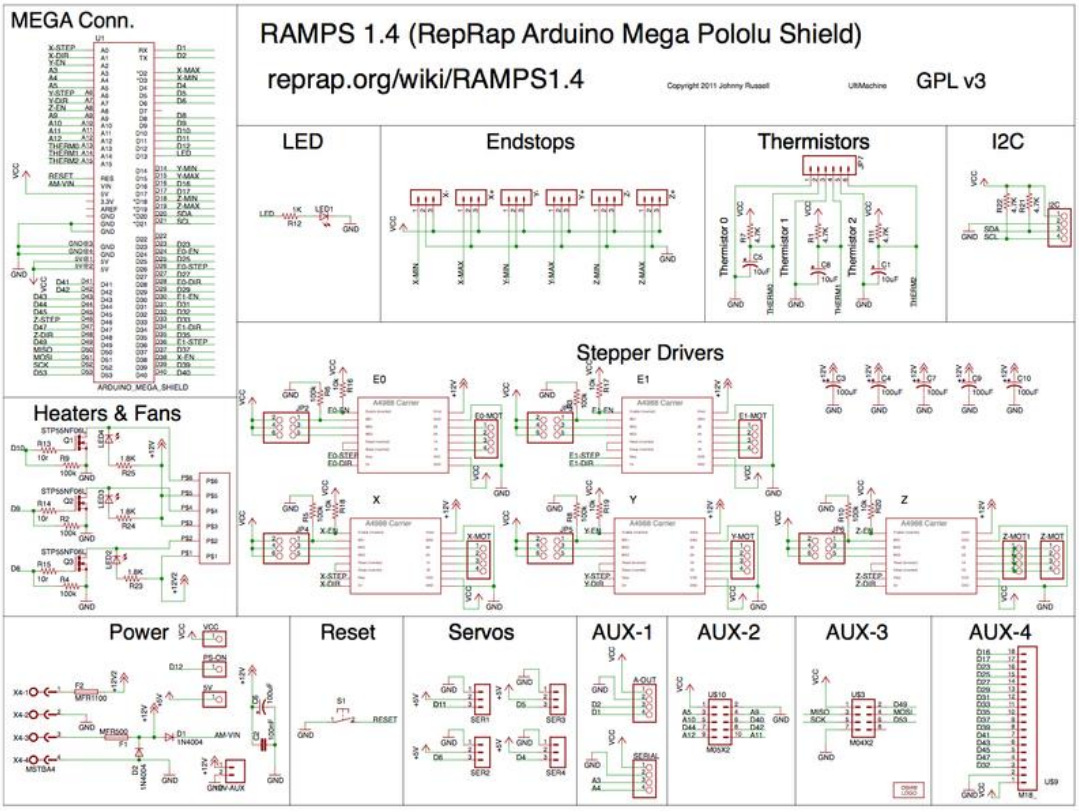
3.6. Hardware

El hardware de la impressora comunament està format per una ramps v1.4 i un arduino i es en el que ens basarem per a realitzar el treball.

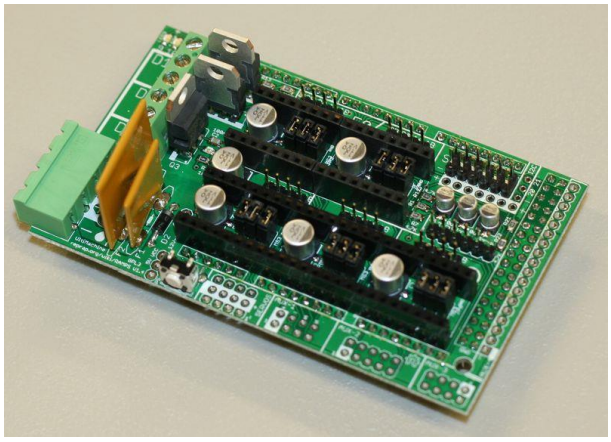
La ramps es la placa de potència encarregada de transformar la senyal que produeix el arduino en una senyal amb suficient voltatge per a poder moure els motors o escalfar el hotend.

La ramps esta formada per transistors, condensadors, resistències, díodes i fusibles. També esta formada per terminals i pins per unir amb elements externs.

El esquema electrònic de la ramps el podem observar mes avall. Com podem veure esta dividida en la part de les connexions amb l'arduino, els endstops, els termistors i el tractament de senyal, els elements que s'escalfen, els controladors dels motors, i el seu tractament, la alimentació, els servos, i la sortida de la pantalla lcd amb la sd, i filament la part dels servos, que en el nostre muntatge no els utilitzarem.



Imatge 12: Esquema elèctric ramps v1.4



Imatge 13: Ramps v1.4

3.7. Software

El software més utilitzat, al ser open source és el Marlin.

Aquest software es basa en controlar les sortides de l'arduino a través d'un arxiu on estan tots els passos a seguir per tots els elements de la impressora per tal de formar l'objecte desitjat. Aquest arxiu està en format gcode.

El format gcode és un llenguatge màquina força semblant al que es fa servir per moure robots. LA filosofia Opensource i RepRap les ha utilitzat per poder programar de forma senzilla els moviments que calen per realitzar les figures. RepRap es una filosofia que es segueix en el món de les impressores 3d on bàsicament a partir de una impressora ja construïda es pugui auto reproduir. Això vol dir que la idea es que totes les peces que formen una impressora siguin imprimibles per ella mateixa. Fa servir una sèrie de sigles per indicar el que es vol realitzar, a continuació estan explicats en una taula les ordes bàsiques, les lletres nnn simbolitzen números:

Lletra	Significat
Gnnn	Comando GCode estàndard, com moure fins a un punt. De forma rectilínia, amb un arc controlat, per moure's fins al punt d'origen directament, per controlar el sensor del eix Z, per conèixer el estat dels endstops, entre d'altres.
Mnnn	Comandament definit per RepRap, com encendre un ventilador, parar o deixar en repòs la impressora, llegir de la targeta sd, realitzar l'escalfament de la impressora, llegir temperatura i llegir posició actual dels eixos.
Tnnn	Seleccionar l'eina nnn. En RepRap, les eines són extrusors
Snnn	Paràmetre de comandament, com la tensió enviada a un motor
Pnnn	Paràmetre de comandament, com el temps en mil·lisegons
Xnnn	Una coordenada X, normalment per moure-hi. Pot ser un nombre enter o racional.
Ynnn	Una coordenada Y, normalment per moure-hi. Pot ser un nombre enter o racional.
Znnn	Una coordenada Z, normalment per moure-hi. Pot ser un nombre enter o racional.

Fnnn	Feedrate en mm per minut. (Velocitat de moviment del capçal d'impressió)
Rnnn	Paràmetre - usat per a temperatures
Ennn	Longitud a extruir en mm. És exactament com X, Y i Z, però per la quantitat de filament a extruir.
Nnnn	Nombre de línia. Utilitzat per demanar la repetició de la transmissió en cas d'errors de comunicació.
*nnn	Checksum. Usat per a comprovar errors de comunicació.

Aquests comandaments que es poden combinar entre ells, per exemple el Gnnn junt amb Xnnn, Ynnn i Znnn, són tractats per el marlin que estarà carregat en el arduino de la impressora. Però no tot es basa en això, també s'encarrega de mostrar la informació en un display. Això vol dir que s'encarrega de processar la informació mentre la executa per poder mostrar informació al usuari.

Es per això que el software, programat amb arduino, no es basa en un simple programa, sinó que està format per un programa principal Marlin.ino, i un conjunt molt ampli de llibreries.

Les llibreries que es fan servir en aquest programa són creades per a cada funció.

Trobem una llibreria on es configuren tots els tipus de Hardwares que es poden utilitzar, una llibreria on estan descrits tots els missatges que pot mostrar el lcd en tot els idiomes per els quals ha estat programada, configura els paràmetres bàsics de la impressora.

La llibreria que conté tota la informació necessària per tractar el gcode és la Marlin_main.cpp i Marlin_main.h

Les segones més importants són les llibreries encarregades de configurar les característiques bàsiques de la impressora, ja que al ser un programa open source, es busca que sigui vàlid per a qualsevol tipus de dispositiu.

I a partir d'aquest punt les llibreries que trobem serveixen per fer càlculs per controlar les sortides del arduino i les entrades, per exemple, per convertir el moviment en centímetres del gcode en un impuls elèctric vàlid per moure els eixos la distancia desitjada.

I així amb la temperatura.

També hi ha unes llibreries importants que serveixen per adaptar els missatges que es volen mostrar al display, si es que es disposa d'un, en els vàlids per les qualitats de la que es té, per exemple la mida, la resolució, el soroll o els colors.

3.8. Interacció amb l'usuari

Per poder interactuar amb la impressora actualment es fan servir dos formes de comunicació diferents, amb fil o sense fils.

Amb la interacció amb la impressora el que es busca es poder imprimir, o poder canviar qualsevol data preestablerta que es vulgui modificar sobre la marxa, ja sigui temperatura, velocitat o posicionar els eixos de forma correcta.

També es pot engegar o aturar una impressió si no esta quedant de forma satisfactòria. També es pot parar momentàniament per si ens hem quedat sense material en mig d'una peça. O també es pot utilitzar per fer jocs amb els colors en mig de una impressió si només es disposa d'un sol extrusor.

3.8.1. Connexió amb fils actual

La connexió amb fils actual es realitza mitjançant dos formes, una pantalla connectada a la ramps v1.4, que connecta directament amb l'arduino a través dels pins corresponents, o a través d'un ordinador connectat al arduino.

Hi ha una gran varietat de pantalles que es poden connectar directament a la ramps v1.4 per fer mes còmode la impressió, ja que així ens estalviem el haver de tenir connectat un ordinador, i per tant que estigui encès, sempre que es vulgui realitzar una impressió. El que és important tenir en compte és si la pantalla està dotada de un lector de targetes sd, sinó no serà valida, ja que no podrem imprimir directament des de la pantalla. Dins de la gran varietat de pantalles que es poden trobar, cal destacar que actualment s'estan incorporant pantalles tàctils.

La connexió amb el ordinador realitza amb un cable USB, i a partir de un altre software podem interactuar amb la impressora sense cap problema. Podem trobar una gran varietat de programes que ens permeten fer aquesta tasca, com per exemple el Repetier Host, o el Printrun.

3.8.2. Connexió sense fils actual

Per poder realitzar la connexió sense fils actualment es precisa de una raspberry o d'un ordinador.

A través de la raspberry podem obtenir una sèrie de avantatges respecte l'ús de l'ordinador, ja que ens proporciona més serveis, com per exemple, i el més evident es la mida, és molt més pràctic afegir una petita raspberry a la impressora, que haver de tenir tot el dia un ordinador connectat amb una pantalla per poder tenir connexió a internet amb la impressora. També ens ofereix la avantatge de poder afegir una petita càmera per poder veure l'objecte mentre s'està imprimir i poder pausar la impressió des de qualsevol puc sempre que es tingui connexió a internet amb un dispositiu.

Aquesta connexió es realitza amb un programa el qual el més famós es el OctoPrint, amb aquest software podem controlar tot el esmenat anteriorment sense cap problema. L'Octoprint, és un software gratuït i lliure capaç de realitzar el control de la impressora, realitzar un seguiment dels paràmetres i mostrar en temps real el percentatge de impressió els moviments dels eixos i les temperatures del extrusor i del llit, això vol dir que també és capaç de modificar aquests paràmetres a mitja impressió i de parar-la i tornar a començar.

Es per això que en aquest TFG intentarem gaudir de els avantatges que es ofereix la raspberry pi connectada a l'arduino, sense necessitat de disposar del segon element.

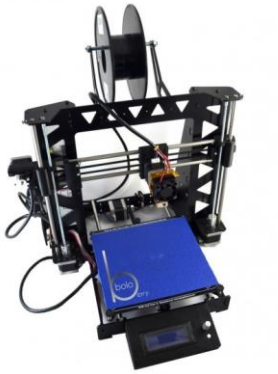
3.9. Model utilitzat per realitzar el treball

En aquest treball utilitzarem un model anomenat Prusa P3 steel pro, aquesta impressora es basa en la tecnologia FFF i de model de eixos cartesians.

Les característiques tècniques son:

- Dimensions de la impressora: 450x430x450 mm
- Extrusor únic
- Llit calent amb relé de protecció
- Vidre de Borosilicat apte per a canvis de temperatura
- Electrònica completament tancada composta per RAMPS 1.4 i Arduino Mega 2560
- Drivers DRV8825 d'alta precisió
- Àrea d'impressió: 200x200x200 mm
- Extrusió Directa Mk8 (Compatible amb materials flexibles)
- Velocitat de moviment màxim: 200 mm / s

- Resolució en l'eix Z: 0,06 mm
- Pantalla LCD 2004
- Software implementat: Marlin



Imatge 14: Prusa P3 steel pro

4. Electrónica

Per a realitzar el nostre objectiu s'ha hagut de redissenyar una ramps v1.4 per tal de poder ser utilitzada per una Raspberry Pi.

S'ha triat la Raspberry Pi model 3, ja que acobla en el seu circuit un chipset Broadcom BCM2387 amb quatre nuclis ARM Cortex-A53 a 1.2 GHz. Aquest processador és capaç de moure amb soltesa videojocs i aplicacions, a més de disposar d'una gran potència de processament per a un altre tipus de tasques. La GPU encarregada dels gràfics és la Broadcom VideoCore IV, una solució Dual Core compatible amb Open GL ÉS 2.0 i OpenVG que permet arribar a resolucions Full HD amb soltesa. Disposa de 4 ports USB. Conté connectivitat Wi-Fi i Bluetooth integrades i una entrada. Té unes dimensions de 85 x 56 x 17 mm. Les especificacions tècniques són:

- Processador:
    - Chipset Broadcom BCM2387.
    - 1,2 GHz de quatre nuclis ARM Cortex-A53
- GPU
    - Dual Core VideoCore IV ® Multimèdia Co-processador. Proporciona Open GL ÉS 2.0, OpenVG accelerat per maquinari, i 1080p30 H.264 d'alt perfil de descodificació.
    - Capaç d'1 Gpixel / s, 1.5Gtexel / so 24 GFlops amb el filtrat de textures i la infraestructura DMA
- RAM: 1GB LPDDR2.
- Connectivitat
    - Ethernet sòcol Ethernet 10/100 BaseT
    - 802.11 b / g / n LAN sense fils i Bluetooth 4.1 (Classic Bluetooth i LI)
    - Sortida de vídeo
        - HDMI rev 1.3 i 1.4
        - RCA compost (PAL i NTSC)
    - Sortida d'àudio

- jack de 3,5 mm de sortida d'àudio, HDMI
- USB 4 x Connector USB 2.0

- Connector de la càmera de 15 pins càmera MIPI interfície en sèrie (CSI-2)
- Pantalla de visualització Connector de la interfície de sèrie (DSI) Connector de 15 vies plana flex cable amb dos carrils de dades i un carril de rellotge
- Ranura de targeta de memòria Empenta / tiri Micro SDIO

4.1. Estudi solució proposta

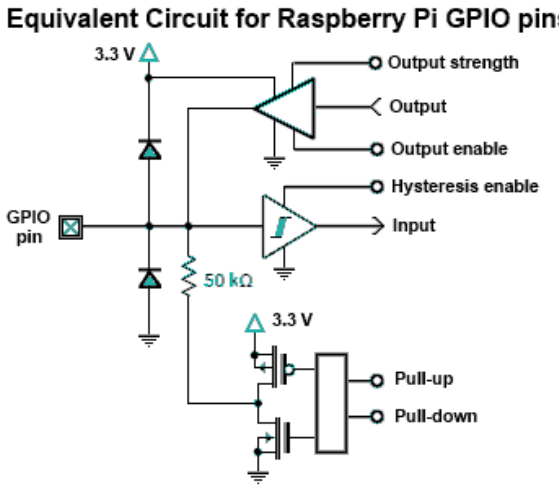
Per poder aconseguir el objectiu, primerament s'ha hagut de fer un estudi de les sortides que es poden utilitzar de la raspberry i quines a característiques s'havien de rebutjar ja que no hi havia sortides suficients.

4.1.1. Característiques GPIO

La Raspberry Pi ofereix els pins d'entrada / sortida digitals de propòsit general (anomenats pins GPIO) que es poden utilitzar per a la lectura de senyals lògiques digitals o per donar sortida als nivells lògics digitals. Les sortides no tenen molta capacitat.

Els pins GPIO de Raspberry Pi són molt versàtils, i es poden modificar moltes de les seves característiques des del programari. Es poden activar / desactivar la histèresi del pin de l'entrada, limitar la velocitat del flux de sortida i controlar la *capacitat d'* accionament de l'alimentació i de l'origen *,* des de 2 mA fins a 16 mA en increments de 2 mA. Aquestes propietats s'estableixen per al bloc GPIO en general, no en funció d'un pin-per-pin.

La capacitat d'actualització d'origen / embornal no limita la corrent cap a fora del pin, però només especifica la intensitat màxima per a la qual es compleixen les especificacions d'alta / baixa tensió de senyal de sortida, la RPi ve amb les sortides GPIO establertes a 8 mA de capacitat.



Imatge 15: Circuit equivalent per a un pin de Raspberry Pi GPIO. Els díodes d'entrada són, en realitat, FET paràsits que no poden manejar cap corrent significativa.

Hi ha algunes subtileses i característiques notables del circuit:

- La porta d'entrada detecta si el nivell de tensió d'entrada és baix o alt comparant-lo amb un voltatge de llindar. Normalment, el llindar de tensió és d'aproximadament 1.8V, però no està garantit; pot estar entre la màxima entrada d'entrada baixa i mínima alta, és a dir, entre uns 0,8 i 2,0 V.

- La sortida condueix a 3.3V (alt) o 0V (baixa). Només es mostra un controlador en l'esquema, però actualment hi ha diversos connectats en paral·lel, amb control de programari habilitats, de manera que la impedància de la unitat de sortida es pot variar per oferir una capacitat actual de 2 mA a 16 mA i una capacitat d'enfonsament, en 2 increments de mA.

Les seves especificacions GPIO es resumeixen a la taula següent:

GPIO d'entrada / sortida de les característiques elèctriques del pin	
Sortida de baixa tensió V_{OL}	<0.40 V
Sortida d'alta tensió V_{OH}	> 2.90 V
Entrada de baixa tensió V_{IL}	<1.15 V
Entrada d'alta tensió V_{IH}	> 2,15 V

Histèresis	0,66 - 2,08 V
Schmitt disparador entrada baix llindar V_{T-}	0.9
Schmitt disparador d'entrada alt llindar V_{T+}	0,90 V
Resistència a l' estirada / baixada	100 KΩ
Corrent de pujada / baixada	<28 uA
Capacitat del pin	5 pF
Resistència a l'espera de l'autobús	5-11 KΩ

Taula 1: Raspberry Pi GPIO especificacions elèctriques de pin de sortida d'entrada

S'ha de tenir en compte les següents especificacions:

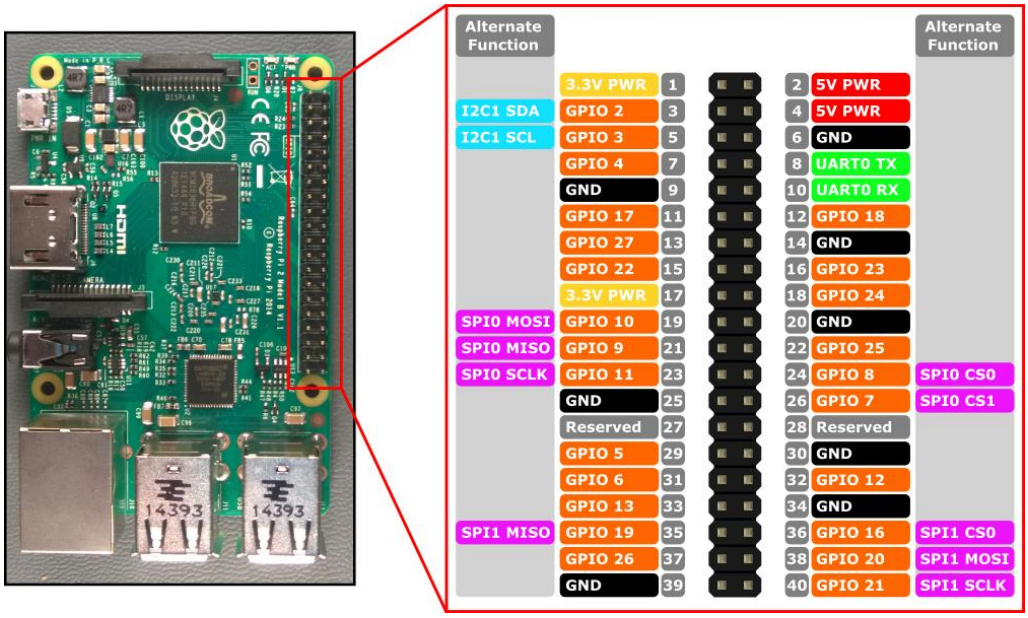
- Aquests són pins de lògica de 3,3 volts. Una tensió propera a 3,3 V s'interpreta com una lògica mentre que una tensió propera a zero volts és una lògica zero. Un pin GPIO mai no s'hauria de connectar a una font de tensió superior a 3.3V o inferior a 0V, ja que pot produir-se un dany ràpid al xip com els díodes de substrat d'entrada de la unitat . És possible que hi hagi moments en què pugui necessitar connectar-los a tensions fora de rang; en aquests casos, el corrent d'entrada ha de ser limitat per una resistència externa a un valor que impedeixi el dany al xip.

- Per evitar una dissipació excessiva de potència en el xip, no s'hauria d'enregistrar / enfonsar més corrent des del pin que el límit programat.

- Mai exigiu que cap font de sortida s'incrementi més de 16 mA.

- El corrent procedent de les sortides es dibuixa a partir del subministrament de 3.3 V, que només pot subministrar un màxim de 50 mA. En conseqüència, el màxim que pot obtenir de totes les sortides de GPIO alhora és inferior a 50 mA.

Els senyals GPIO i fonts d'alimentació es presenten en un encapçalament de 26 pins, proporciona dues files de pins, imparells i parells.

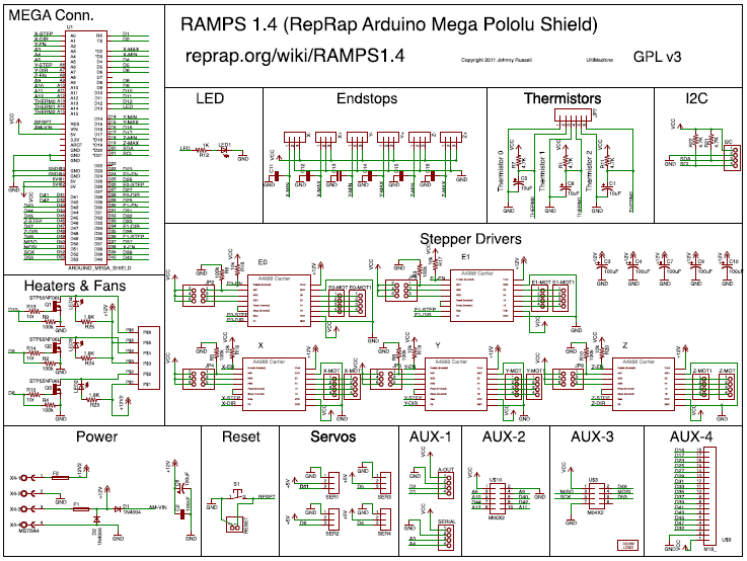


Imatge 16: Disposició GPIO

4.1.2. Característiques seleccionades

Per tant, tenint en compte les sortides disponibles amb les quals interactuar, que són 26, s'han hagut de seleccionar les propietats bàsiques necessàries per poder ficar en funcionament la impressora 3d que hem seleccionat.

- Cada motor necessita tres pins per poder funcionar amb els driver, per tant, s'han mantingut 4 Drivers amb que moure els 4 eixos (x, y, z, e), però en el eix z aniran dos motors connectats sèrie.
- S'han mantingut les entrades dels endstops només per els punts mínims, ja que són els punts on van connectats els nostres sensor.
- S'han mantingut dos sortides d'alimentació de les resistències encarregades de escalfar el extrusor i el llit calent.
- D'igual manera, s'han mantingut dos entrades de mesura de temperatura, però en aquest cas es precisarà utilitzar sensor digitals.
- S'ha mantingut la sortida del LED per conèixer si la electrònica esta alimentada de forma correcta, igual que s'ha mantingut el botó de reset i la entrada de alimentació a la raspberry pi i a tot el circuit que el conforme.
- Totes les altres característiques de la ramps original han sigut descartades per falta de pins amb els que rebre i tractar informació.
- S'ha pogut eliminar la connexió amb la pantalla LCD ja que la raspberry disposa d'una sortida directe de HDMI.



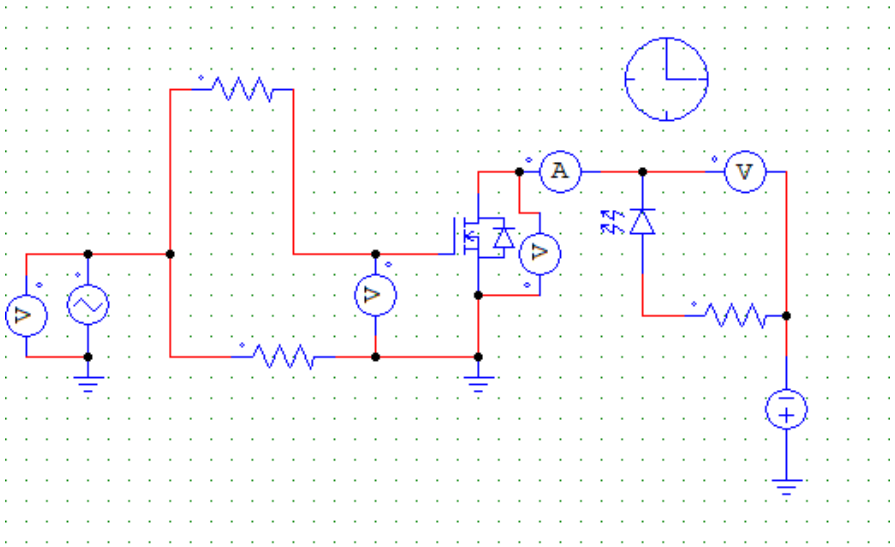
Imatge 17: Ramps v1.4 original

4.1.3. Adaptació circuit

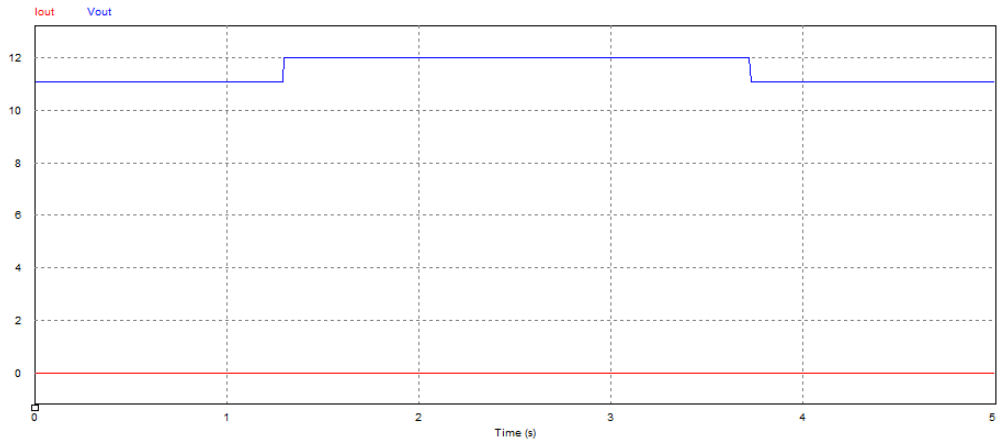
Per poder utilitzar la raspberry com a codificador de la informació s'ha hagut de tenir especial cura en les característiques de les entrades i sortides per tal de no malmetre el aparell.

Per tant s'han realitzat comprovacions a través del programa PSIM que es mostraran a continuació.

Per a comprovar les sortida de la alimentació de les parts que s'han d'escalfar:

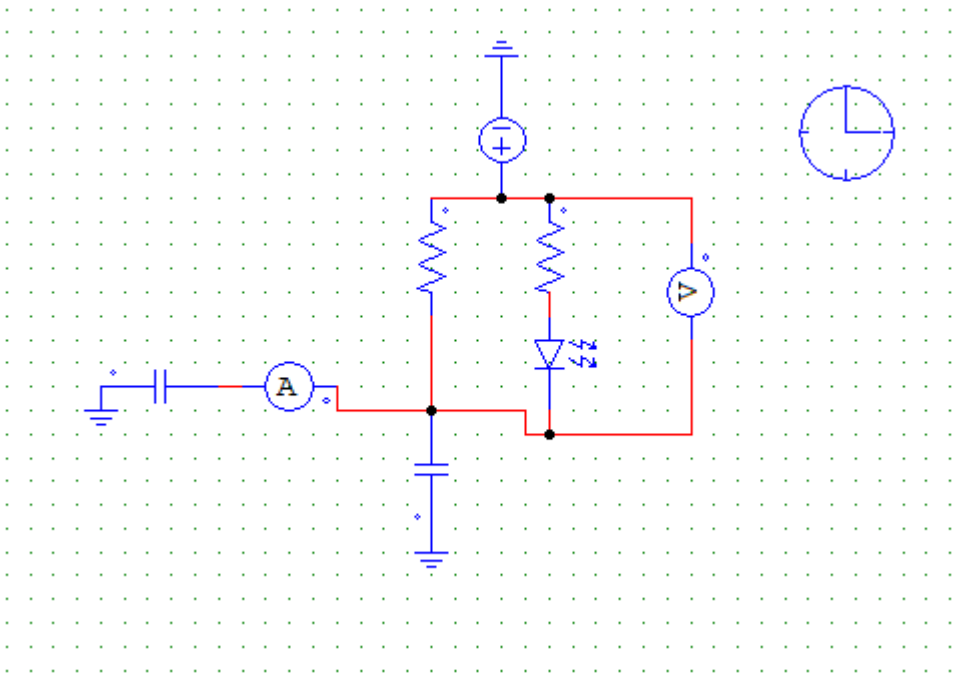


Imatge 18: Esquema elèctric circuit encarregat d'escalfar el extrusor i el llit calent



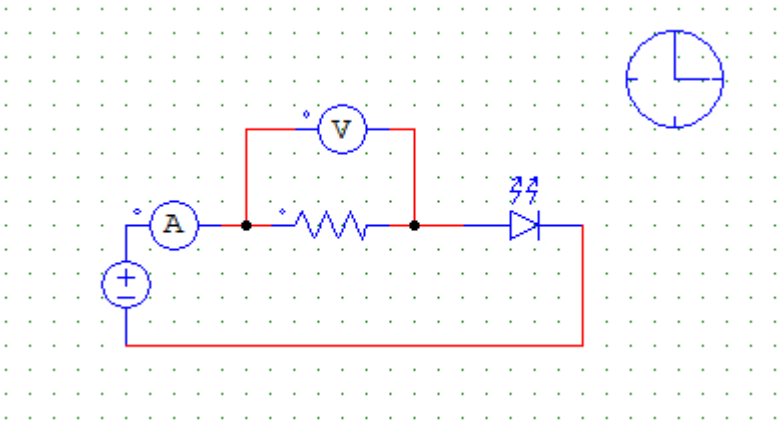
Imatge 19: Resposta eléctrica

Per a comprovar la sortida dels finals de carrera:



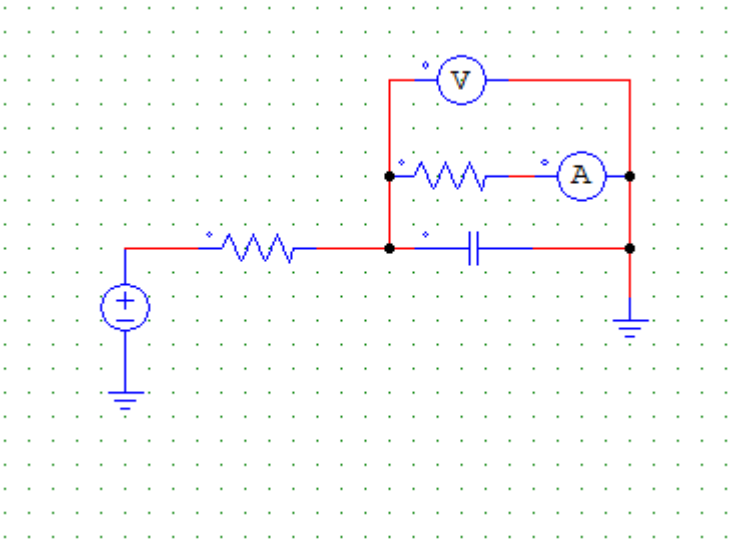
Imatge 20: Circuit elèctric dels finals de carrera

Per comprovar el LED de encesa de la raspberry:



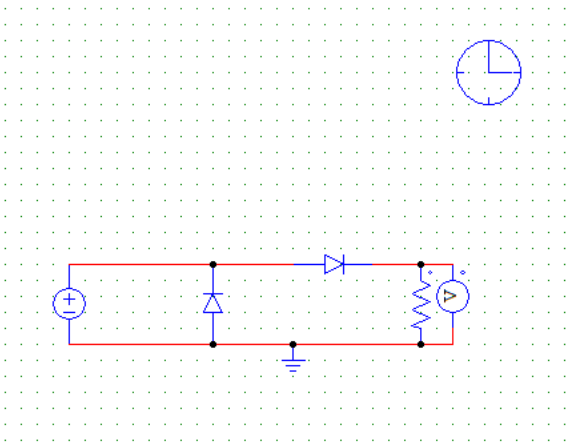
Imatge 21: Circuit elèctric del LED d'encesa

Simulació de les senyals del termistor:



Imatge 22: Circuit elèctric dels senyals dels termistors

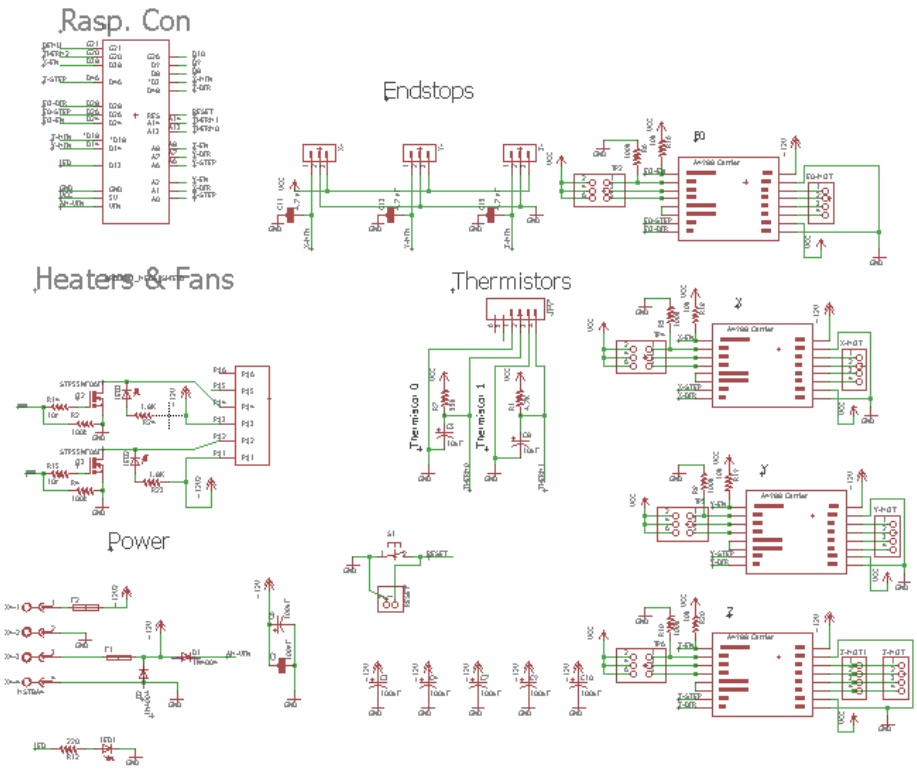
I per finalitzar la simulació de la alimentació de la raspberry pi a partir de la Raspra:



Imatge 23: Circuit elèctric de alimentació de la Raspberry Pi a partir de la Raspra

Per a realitzar aquesta tasca s'han hagut de conèixer les propietats dels elements externs que s'estaven utilitzant, com els termistors, la resistència ceràmica, els finals de carrera mecànics.

Per tant, finalment, el esquema electric final de la Raspra ha sigut així:



Imatge 24: Esquema Raspra amb els components bàsics seleccionats

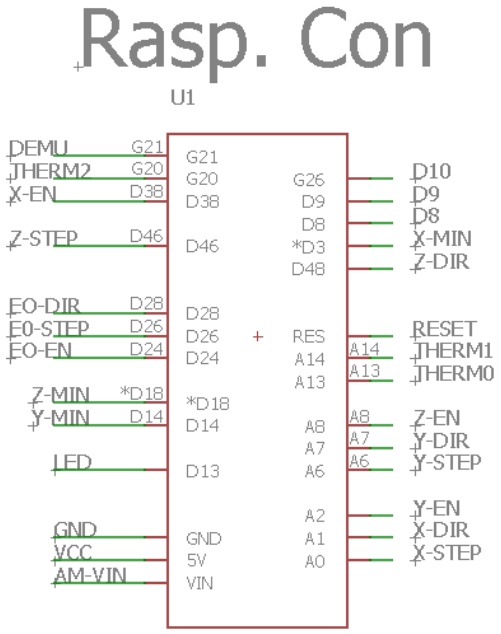
4.2. Disseny PCB

Per dissenyar la PCB ens hem basat en la ramps v1.4 i en les característiques esmentades anteriorment.

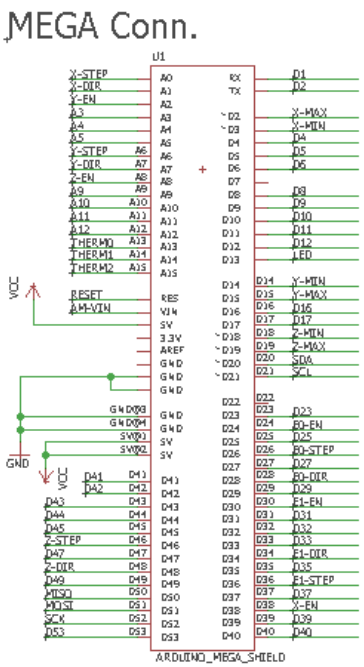
Un cop seleccionats els circuits i els components que formaran la nostre PCB, s'ha realitzat el esquema real.

Per a realitzar aquest procés hem utilitzat el programa EAGLE, per tant un com teníem el circuit dibuixat de forma esquemàtica. Si teníem ben seleccionats els packaging. En el nostre cas em volgut utilitzar elements SMD en els que era convenients per tan de optimitzar l'espai.

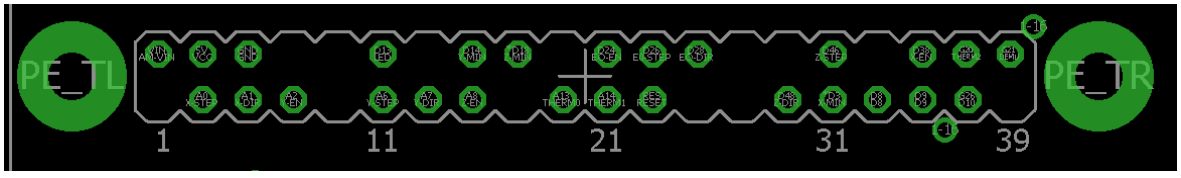
Hem hagut de redissenyar dos elements. En aquest cas el packaging del arduino mega per tal de convertir-lo en la raspberry amb les mateixes mesures i sortides.



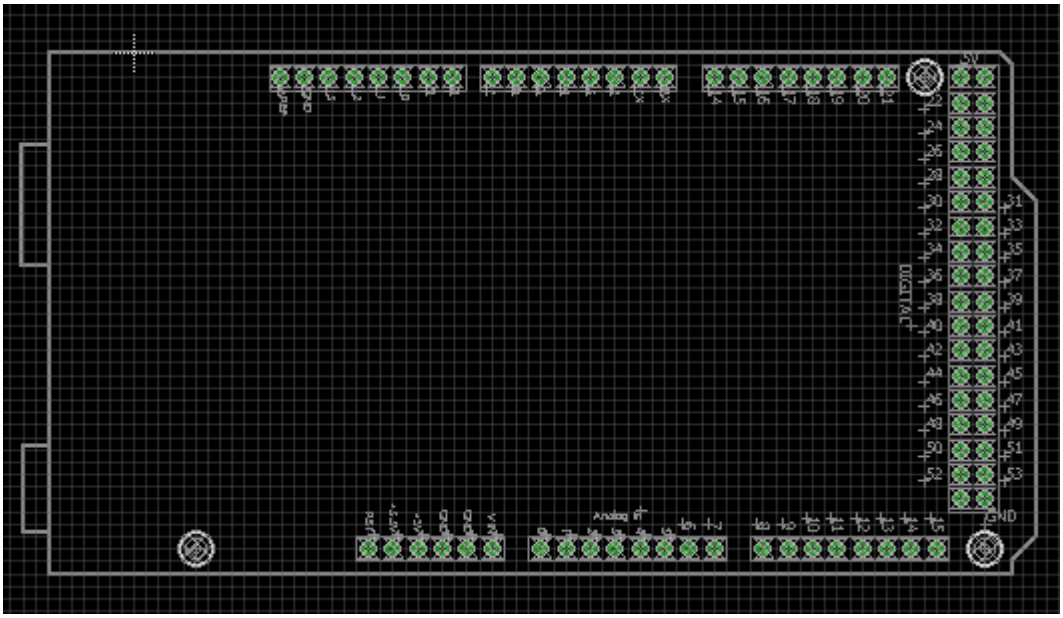
Imatge 26: Símbol Raspberry en EAGLE



Imatge 25: Símbol Arduino MEGA



Imatge 27: Packaging Raspberry Pi adaptat a les necessitats

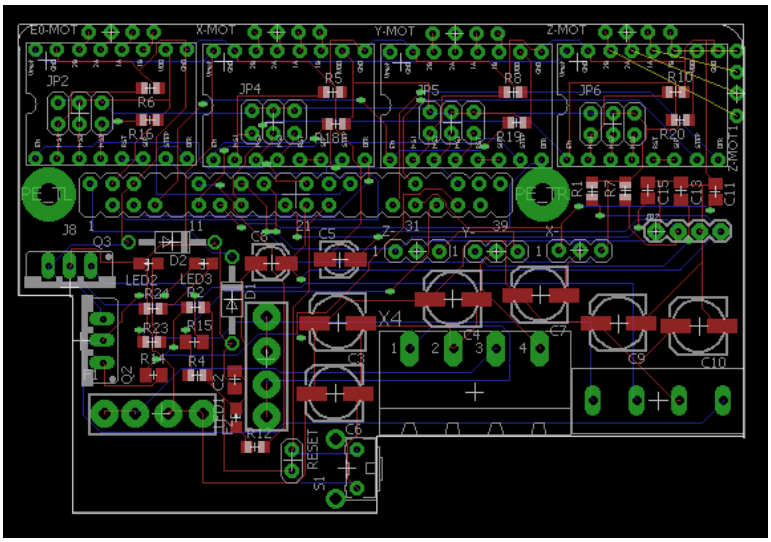


Imatge 28: Packaging Arduino MEGA

4.3. Característiques

El que s'ha hagut de tenir en compte per a poder realitzar les PCB han sigut les limitacions que ens proporcionava la eina que faríem servir per realitzar la PCB. En aquest cas el mètode d'insolació que ens proporcionava la universitat.

Per tant, les pistes havien de tenir un mida mínima, una separació mínima entre pistes, un diàmetre de pads mínim de 0.5mm. Per això s'ha hagut de redissenyar el circuit diverses vegades fins aconseguir les característiques necessàries.

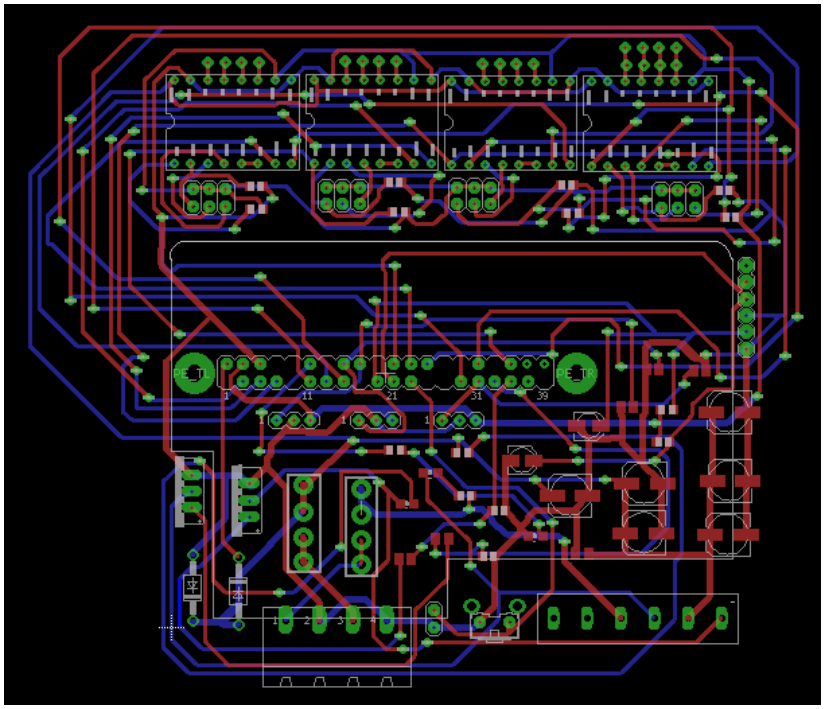


Imatge 29: Primer disseny raspra

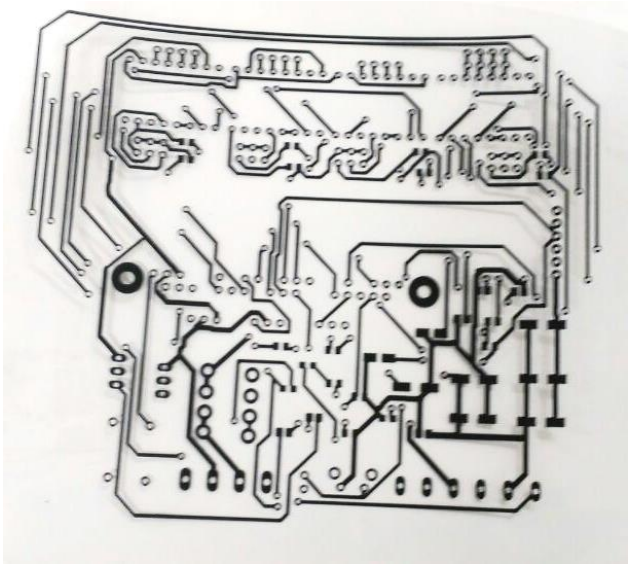
La llista de materials per a realitzar el circuit apareix en el Annex A: Components PCB.

4.4. Disseny final

El disseny final de la raspra el qual en ha permès realitzar de forma física el nostre circuit a sigut el de la Imatge 30.



Imatge 30: Disseny final Raspra



Imatge 31: Fotolit preparat per la insolació de la placa fotosensible

5. Solució Software

5.1. Llenguatge escollit

El llenguatge que s'ha escollit per tal de realitzar el nostre treball ha sigut el python, ja que és el llenguatge natiu de la raspberry, i tot i que sigui mes ineficient, en el nostre cas no es important ja que la velocitat de processament esta condicionada per el moviment físic del motors o la velocitat d'escalfament de l'extrusor.

Aquest llenguatge ha comportat certes dificultats a l'hora de realitzar el software per poder fer servir la impressora.

Per realitzar el software ens hem basat en el software Marlin que està dissenyat amb llenguatge arduino i, per tant, el primer que hem hagut de fer ha sigut entendre com funciona el Marlin i el llenguatge Arduino, ja que no es disposava de un gran coneixement.

El software Marlin esta basat en un conjunt de llibreries, que es arduino es programen a partir de dos arxius diferents, el primer, el cpp és l'arxiu que conté tota la informació necessària per poder utilitzar la llibreria, bàsicament és on està tot el codi important, i el h és on es programa la seqüencia que realitza la llibreria quan es nombrada, com si fos un subprograma.

Per tant per programar el nostre programa en python primer hem hagut de fer una selecció de quines eren les llibreries que no faríem servir. Finalment ens hem quedat amb: Marlin.h, Marlin_main.cpp, configuration, configuration_adv, configurationStore, Language, motion_control, pins_raspberry, planner, steppers, temperature, Thermistorables, i també s'han utilitzat llibreries pròpies del software, com math, GPIO, spi, fastio.

Hem pogut descartar molts elements, ja que la nostre placa és única, només hem seleccionat un idioma, no hem incorporat pantalla lcd ni lectura de sd. La lectura del arxiu es farà a través del programa OctoPrint.

Seguidament es va procedir a traduir les llibreries de Arduino a Python, per poder realitzar aquest pas primerament s'ha hagut de aprendre el llenguatge Python per poder realitzar aquest pas. L'aprenentatge s'ha realitzat a partir del llibre "*Aprenda a Pensar Como un Programador"* (Downey, A., Elkner, J. i Meyers, C. 2002), i seguidament s'ha començat a fer la traducció.

5.1.1. Diferències trobades

En aquest pas s'ha trobat un seguit de problemes, ja que hi ha comandaments de Arduino que no es poden realitzar a python com per exemple el switch-case, que per a poder realitzar aquest comandament s'ha utilitzat el try.

```
else if(code_seen('M'))
{
  switch( (int)code_value() )
  {
#ifdef ULTIPANEL
  case 0: // M0 - Unconditional stop - Wait for user button press on LCD
  case 1: // M1 - Conditional stop - Wait for user button press on LCD
  {
    char *src = strchr_pointer + 2;

    codenum = 0;

    bool hasP = false, hasS = false;
    if (code_seen('P')) {
      codenum = code_value(); // milliseconds to wait
      hasP = codenum > 0;
    }
    if (code_seen('S')) {
      codenum = code_value() * 1000; // seconds to wait
      hasS = codenum > 0;
    }
    starpos = strchr(src, '*');
    if (starpos != NULL) *(starpos) = '\0';
```

Imatge 32: Exemple switch-case Arduino

```
def case351(code): # M351 Toggle MS1 MS2 pins directly, S# determines MS1 or MS2, X# sets the pin high/low.
    if raspra.pins_raspberry.X_MS1_PIN > -1:
        if code_seen('S'):
            raspra.stepper.microstep_readings()
            try:
                ({'1':x1,'2':x2}[code_value(code)])()
            finally:
                print('Error')
            return

def case999(): #M999: Restart after being stopped
    Stopped = False
    # lcd_reset_alert_level()
    gcode_LastN = Stopped_gcode_LastN
    FlushSerialRequestResend()
    return

caseG={'0':case0,'1':case1,'2':case2,'3':case3,'4':case4,'11':case11,'28':case28,
    '31':case31,'32':case32,'90':case90,'91':case91,'92':case92}

caseM = {'20':case20,'21':case21,'22':case22,'23':case23,'24':case24,
      '25':case25,'26':case26,'27':case27,'28':case28,'29':case29,
      '30':case30,'32':case32,'928':case928,'31':case31,'42':case42,
```

Imatge 33: Exemple Try en Python

```python
def process_commands(code):
    starpos = None
    if code_seen('G'):
        try:
            (caseG[code_value(code)])()
        finally:
            print("Error")
    elif code_seen('M'):
        if code_value(code)==17:
            # LCD_MESSAGEPGM(MSG_NO_MOVE)
            raspra.Marlin_h.enable_x()
            raspra.Marlin_h.enable_y()
            raspra.Marlin_h.enable_z()
            raspra.Marlin_h.enable_e0()
```

Imatge 34: Segon exemple Try en Python

Una altre diferencia es el comando delay(1000);, en python és: time.sleep (1000 / 1000.0).

També s'ha de tenir en compte les diferències bàsiques com els finals de línia, que en arduino es un punt i coma i en python no hi ha final de frase, i en comptes de obrir i tancat parèntesis per definir les funcions, es realitza amb el espais.

On s'ha trobat dificultat ha sigut amb els for, ja que són totalment diferents:

```
sum=0.0;
for( j=0; j<=n; j++) {
  sum = sum + sample_set[j];
}
mean = sum / (double (n+1));
```

Imatge 35: Exemple For en Arduino

```python
for i in raspra.configuration.NUM_AXIS:
  current_position[i] = destination[1]
previous_millis_cmd = time.time() * 1000.0
```

Imatge 36: Exemple For en Python

En la Imatge 36: Exemple For en Python, podem observar com es crida a un element definit en el mòdul configuration dins de la carpeta raspra que és on hem guardar tots els mòduls.

Això també és una gran diferència, ja que en Arduino només es necessita importar la llibreria al principi del codi si es vol utilitzar una variable declarada en aquella llibreria , en canvi en Python quan es vol utilitzar una variable d'una llibreria, s'ha d'importar al principi del programa, però cada cop que es nombri s'ha de traçar el camí en punts.

```
#include "ultralcd.h"
#include "planner.h"
#include "stepper.h"
#include "temperature.h"
#include "motion_control.h"
#include "cardreader.h"
#include "watchdog.h"
#include "ConfigurationStore.h"
#include "language.h"
#include "pins_arduino.h"
#include "math.h"
```

Imatge 37: Importació llibreries Arduino

```
import time

import math
import sys
import serial
import spi
import RPi.GPIO as GPIO

import raspra.ConfigurationStore
import raspra.Marlin_h
import raspra.configuration
import raspra.configuration_adv
import raspra.language
import raspra.marlinmod.fastio
import raspra.mation_control
import raspra.pins_raspberry
import raspra.planner
import raspra.stepper
import raspra.temperature
```

Imatge 38: Importació llibreries Python

5.2. Programa principal

El programa principal es el Marlin_main, en aquest mòdul és on es llegeix el gcode i es realitzen les accions de cada comandament.

Per fer això es basa en un try, amb el qual a partir de la primera lletra i del número que el segueix s'activa una funció que realitza una acció sobre els motors o els termistors de la impressora.

Per tant, des de el Marlin_main es criden els altres mòduls on estan programades les funcions que necessita el Marlin_main per executar les acciones de forma correcta.

A continuació es veurà un fragment del cos del codi:

```
def case0():
    return
def case1(code, current_position, target, feedmultiply):
    if (Stopped == False):
        get_coordinates(code)  # For X Y Z E F
        prepare_move(current_position, feedmultiply)
    return
def case2(code,target,feedmultiply):# G2  - CW ARC
    if (Stopped == False):
        get_arc_coordinates(target, code)
        prepare_arc_move(True, feedmultiply)
    return
def case3(code,target, feedmultiply):# G3  - CCW ARC
    if (Stopped == False):
        get_arc_coordinates(target, code)
        prepare_arc_move(False,feedmultiply )
    return

def case4(code, gcode_LastN):# G4 dwell
    # LCD_MESSAGEPGM(MSG_DWELL)
    codenum = 0
    if (code_seen('P')):
        codenum = code_value(code)  # milliseconds to wait
    if (code_seen('S')):
        codenum = code_value(code)*1000  # seconds to wait
        raspra.stepper.st_synchronize()
    codenum += time.time() * 1000.0  # keep track of when we started waiting
    previous_millis_cmd = time.time() * 1000.0
    while (time.time() * 1000.0 < codenum):
        raspra.temperature.manage_heater()
        manage_inactivity(gcode_LastN, code)
```

```
        manage_inactivity(gcode_LastN, code)
        # lcd_update()
    return previous_millis_cmd

def case11(): # G11 retract_recover
    return

def case28(code, current_position,strchr_pointer):#G28 Home all Axis one at a time
    global destination
    feedrate = 1500.0
    saved_feedrate = feedrate
    feedmultiply = 100
    saved_feedmultiply = feedmultiply
    previous_millis_cmd = time.time() * 1000.0
    raspra.stepper.enable_endstops(True)
    for i in current_position :
        destination = current_position[i]
        feedrate = 0.0
    home_all_axis = not((code_seen(code)))
    if raspra.configuration.Z_HOME_DIR > 0 :  # If homing away from BED do Z first
        if((home_all_axis) or (code_seen(axis_codes[2]))):
            homeaxis(2)

    plan_set_position=(current_position[0], current_position[1], current_position[2],current_position[3])
    destination[0] = 1.5 * raspra.configuration.X_MAX_LENGTH* x_axis_home_dir
    destination[1] = 1.5 * max_length[1] * home_dir[1]
    feedrate = homing_feedrate[0]
    if homing_feedrate[1] < feedrate:
      feedrate = homing_feedrate[1]
```

```
    else:
      feedrate = feedrate+(math.sqrt(pow(max_length[0] / max_length[1], 2) + 1))
    plan_buffer_line=(destination[0], destination[1], destination[2], destination[3], feedrate / 60, active_extruder)
    raspra.stepper.st_synchronize()

    plan_set_position=(current_position[0], current_position[1], current_position[2], current_position[3])
    destination[0] = current_position[1]
    destination[0] = current_position[1]
    plan_buffer_line=(destination[0], destination[1], destination[2], destination[3], feedrate / 60,active_extruder)
    feedrate = 0.0
    raspra.stepper.st_synchronize()
    raspra.stepper.endstops_hit_on_purpose()

    current_position = [destination[0],destination[1],destination[2]]

    if home_all_axis or code_seen(axis_codes[0]):
      homeaxis(0)

    if home_all_axis or code_seen(axis_codes[1]):
      homeaxis(1)

    if (code_seen(axis_codes[0])):
      if (code_value_long(strchr_pointer, cmdbuffer) != 0):
        current_position[0]=code_value(code)+add_homing[0]

    if(code_seen(axis_codes[1])):
        if(code_value_long(strchr_pointer, cmdbuffer) != 0):
            current_position[1]=code_value(code)+add_homing[1]

    if raspra.configuration.Z_HOME_DIR < 0 :  # If homing towards BED do Z last
          if((home_all_axis) or (code_seen(axis_codes[2]))):
            homeaxis(2)
```

```
    return feedrate

def case31(): # dock the sled
    # dock_sled(True)
    return
def case32(): # undock the sled
    # dock_sled(False)
    return

def case90():# G90
      relative_mode = False
      return

def case91():# G91
      relative_mode = True
      return

def case92(code):# G92
      if not code_seen(axis_codes[3]):
        raspra.stepper.st_synchronize()
      for i in raspra.configuration.NUM_AXIS:
        if code_seen(axis_codes[i]):
          if i == 3:
            current_position[i] = code_value(code)
            plan_set_e_position=(current_position[3])
          else:
             current_position[i] = code_value(code)+add_homing[i]
             plan_set_position=(current_position[0], current_position[1], current_position[2], current_position[3])
        return
```

```
caseG={'0':case0,'1':case1,'2':case2,'3':case3,'4':case4,'11':case11,'28':case28,
      '31':case31,'32':case32,'90':case90,'91':case91,'92':case92}

caseM = {'20':case20,'21':case21,'22':case22,'23':case23,'24':case24,
         '25':case25,'26':case26,'27':case27,'28':case28,'29':case29,
         '30':case30,'32':case32,'928':case928,'31':case31,'42':case42,
         '104':case104,'112':case112,'140':case140,'105':case105,'109':case109,
         '81':case81,'82':case82,'83':case83,'18':case18,'84':case84,'85':case85,
         '92':case92,'115':case115,'117':case117,'114':case114,'120':case120,'121':case121,
         '119':case119,'200':case200,'201':case201,'202':case202,'204':case204,'205':case205,
         '206':case206,'220':case220,'221':case221,'226':case226,'300':case300,
         '301':case301,'304':case304,'240':case240,'250':case250,'302':case302,
         '303':case303,'400':case400,'500':case500,'501':case501,'502':case502,
         '503':case503,'600':case600,'907':case907,'908':case908,'350':case350,
         '351':case351,'999':case999}

def process_commands(code):
  starpos = None
  if code_seen('G'):
      try:
          (caseG[code_value(code)])()
      finally:
          print("Error")
  elif code_seen('M'):
       if code_value(code)==17:
           # LCD_MESSAGEPGM(MSG_NO_MOVE)
           raspra.Marlin_h.enable_x()
           raspra.Marlin_h.enable_y()
           raspra.Marlin_h.enable_z()
           raspra.Marlin_h.enable_e0()
```

```
   elif(code_seen('T')):
    tmp_extruder = code_value(code)
    if(tmp_extruder >= raspra.configuration.EXTRUDERS) :
      raspra.Marlin_h.SERIAL_ECHO("T")
      raspra.Marlin_h.SERIAL_ECHO(tmp_extruder)
      raspra.Marlin_h.SERIAL_ECHOLN(raspra.language.MSG_INVALID_EXTRUDER)
    else:
      make_move = False
      if(code_seen('F')) :
        make_move = True
        next_feedrate = code_value(code)
        if(next_feedrate > 0.0):
          feedrate = next_feedrate

      raspra.Marlin_h.SERIAL_ECHO(raspra.language.MSG_ACTIVE_EXTRUDER)
      raspra.Marlin_h.SERIAL_PROTOCOLLN(active_extruder)

  else:
    raspra.Marlin_h.SERIAL_ECHOPGM(raspra.language.MSG_UNKNOWN_COMMAND)
    raspra.Marlin_h.SERIAL_ECHO(cmdbuffer[bufindr])
    raspra.Marlin_h.SERIAL_ECHOLNPGM("\"")

    ClearToSend()


lastMotor = 0 #Save the time for when a motor was turned on last
lastMotorCheck = 0
```

5.3. Llibreries

En aquest punt explicarem per a que serveix cada llibreria, que en python són mòduls.

El primer mòdul que s'explicarà serà el Marlin.h, aquest s'encarrega de executar el Marlin_main, i en ell es declaren variables que es faran servir en el Marlin_main, només es mostrarà alguna part del codi, la més important:

```
ser=serial.Serial('/dev/COM3')

axis_known_position=[1,1,1]

def SERIAL_PROTOCOL(x):
        ser.write(x)
def SERIAL_PROTOCOL_F(x,y):
        ser.write(x)
        ser.write(y)
def SERIAL_PROTOCOLPGM(x):
        serialprintPGM(PSTR(x))
def SERIAL_PROTOCOLLN(x):
        ser.write(x)
        ser.write('\n')
def SERIAL_PROTOCOLLNPGM(x):
        serialprintPGM(x)
        ser.write('\n')

def SERIAL_ERROR(x):
    SERIAL_PROTOCOL(x)
def SERIAL_ERRORPGM(x):
    SERIAL_PROTOCOLPGM(x)
def SERIAL_ERRORLN(x):
    SERIAL_PROTOCOLLN(x)
def SERIAL_ERRORLNPGM(x):
        SERIAL_PROTOCOLLNPGM(x)

def SERIAL_ECHO(x):
        SERIAL_PROTOCOL(x)
def SERIAL_ECHOPGM(x):
        SERIAL_PROTOCOLPGM(x)
def SERIAL_ECHOLN(x):
        SERIAL_PROTOCOLLN(x)
```

```
raspra.Marlin_main.get_command()
raspra.Marlin_main.process_commands()

raspra.Marlin_main.manage_inactivity()

if  raspra.pins_raspberry.X_ENABLE_PIN > -1:
    def enable_x():
        GPIO.output(raspra.pins_raspberry.X_ENABLE_PIN, raspra.configuration.X_ENABLE_ON)
        while (0):
            return
    def disable_x():
        GPIO.output(raspra.pins_raspberry.X_ENABLE_PIN,not raspra.configuration.X_ENABLE_ON)
        axis_known_position[0] = False
else:
    def enable_x():
        return
    def disable_x():
        return

if raspra.pins_raspberry.Y_ENABLE_PIN > -1:
    def enable_y() :
        GPIO.output(raspra.pins_raspberry.Y_ENABLE_PIN, raspra.configuration.Y_ENABLE_ON)
    def disable_y():
        GPIO.output(raspra.pins_raspberry.Y_ENABLE_PIN,not raspra.configuration.Y_ENABLE_ON)
        axis_known_position[1] = False
else:
    def enable_y():
        return
    def disable_y():
        return
```

```
if raspra.pins_raspberry.E0_ENABLE_PIN>-1:
    def enable_e0():
        GPIO.output(raspra.pins_raspberry.E0_ENABLE_PIN, raspra.configuration.E_ENABLE_ON)
    def disable_e0():
        GPIO.output(raspra.pins_raspberry.E0_ENABLE_PIN,raspra.configuration.E_ENABLE_ON)
else:
    def enable_e0():
        return
    def disable_e0():
        return


def AxisEnum():
    X_AXIS=0
    Y_AXIS=1
    Z_AXIS=2
    E_AXIS=3
    X_HEAD=4
    Y_HEAD=5
    return X_AXIS,Y_AXIS,Z_AXIS,E_AXIS,X_HEAD,Y_HEAD

raspra.Marlin_main.FlushSerialRequestResend()
raspra.Marlin_main.ClearToSend()
```

Seguidament detallarem configuration i configuration_adv, en aquests dos mòduls es configuren els paràmetres característics de cada impressora com el número de extrusors, la superfície d'impressió, el tipus de mecànica, els controladors de motors utilitzats, el llenguatge, entre altres conceptes:

Primerament, es mostrarà una part del mòdul configuration:

```
CUSTON_PRUSA_NAME = "MERY"
EXTRUDERS = 1
POWER_SUPPLY = 1

# THERMAL SETTINGS

TEMP_SENSOR_0 = 1
TEMP_SENSOR_BED = 1
MAX_REDUNDANT_TEMP_SENSOR_DIFF = 10
TEMP_RESIDENCY_TIME = 10  # SECONDS
TEMP_HYSTERESIS = 3
TEMP_WINDOW = 1

# The minimal temperature defines the temperature below which the heater will not be enabled It is used
# To check that the wiring to the thermistor is not broken.
# Otherwise this would lead to the heater being powered on all the time.

HEATER_0_MINTEMP = 5
BED_MINTEMP = 5

#  When temperature exceeds max temp, your heater will be switched off.
#  This feature exists to protect your hotend from overheating accidentally, but *NOT* from thermistor short/failure!
#  You should use MINTEMP for thermistor short/failure protection.

HEATER_0_MAXTEMP = 275
BED_MAXTEMP = 150


# PID settings:
# Comment the following line to disable PID and enable bang-bang.
```

```
BANG_MAX = 255  # limits current to nozzle while in bang-bang mode; 255=full current
PID_MAX = BANG_MAX  # limits current to nozzle while PID is active (see PID_FUNCTIONAL_RANGE below);
# 255=full current
PID_FUNCTIONAL_RANGE = 10
PID_INTEGRAL_DRIVE_MAX = PID_MAX
K1 = 0.95
PID_dT = ((raspra.Thermistorables.OVERSAMPLENR * 10.0) / (F_CPU / 64.0 / 256.0))  # sampling period of the
# temperature routine

DEFAULT_Kp = 7.0
DEFAULT_Ki = 1.08
DEFAULT_Kd = 114

# Bed Temperature Control
MAX_BED_POWER = 255

# this prevents dangerous Extruder moves, i.e. if the temperature is under the limit
# can be software-disabled for whatever purposes by

PREVENT_DANGEROUS_EXTRUDE = 1
PREVENT_LENGTHY_EXTRUDE = 1

# Travel limits after homing
X_MAX_POS = 205
X_MIN_POS = 0
Y_MAX_POS = 205
Y_MIN_POS = 0
Z_MAX_POS = 200
Z_MIN_POS = 0

X_MAX_LENGTH = (X_MAX_POS - X_MIN_POS)
Y_MAX_LENGTH = (Y_MAX_POS - Y_MIN_POS)
Z_MAX_LENGTH = (Z_MAX_POS - Z_MIN_POS)
```

A continuació detallarem una part del codi de configuration_adv:

```
# THERMAL SETTINGS

BED_CHECK_INTERVAL = 5000  # ms between checks in bang-bang control

# this adds an experimental additional term to the heating power, proportional to the extrusion speed.
# if Kc is chosen well, the additional required power due to increased melting should be compensated.
PID_ADD_EXTRUSION_RATE = 1
DEFAULT_Kc = 1

AUTOTEMP = 1
AUTOTEMP_OLDWEIGHT = 0.98

EXTRUDER_RUNOUT_MINTEMP = 190
EXTRUDER_RUNOUT_SECONDS = 30.
EXTRUDER_RUNOUT_ESTEPS = 14.  # mm filament
EXTRUDER_RUNOUT_SPEED = 1500.  # extrusion speed
EXTRUDER_RUNOUT_EXTRUDE = 100

TEMP_SENSOR_AD595_OFFSET = 0.0
TEMP_SENSOR_AD595_GAIN = 1.0

CONTROLLERFAN_PIN = -1  # Pin used for the fan to cool controller (-1 to disable)
CONTROLLERFAN_SECS = 60  # How many seconds, after all motors were disabled, the fan should run
CONTROLLERFAN_SPEED = 255  # == full speed

EXTRUDER_0_AUTO_FAN_PIN = -1
EXTRUDER_AUTO_FAN_TEMPERATURE = 50
EXTRUDER_AUTO_FAN_SPEED = 255
```

```
ENDSTOPS_ONLY_FOR_HOMING = 1

X_HOME_RETRACT_MM = 5
Y_HOME_RETRACT_MM = 5
Z_HOME_RETRACT_MM = 2

AXIS_RELATIVE_MODES = {False, False, False, False}
MAX_STEP_FREQUENCY = 40000
INVERT_X_STEP_PIN = False
INVERT_Y_STEP_PIN = False
INVERT_Z_STEP_PIN = False
INVERT_E_STEP_PIN = False

DEFAULT_STEPPER_DEACTIVE_TIME = 60
DEFAULT_MINIMUMFEEDRATE = 0.0  # minimum feedrate
DEFAULT_MINTRAVELFEEDRATE = 0.0

# minimum time in microseconds that a movement needs to take if the buffer is emptied.
DEFAULT_MINSEGMENTTIME = 20000

# If defined the movements slow down when the look ahead buffer is only half full
SLOWDOWN = 1

MINIMUM_PLANNER_SPEED = 0.05  # (mm/sec)

# MS1 MS2 Stepper Driver Microstepping mode table
MICROSTEP1 = LOW, LOW
MICROSTEP2 = HIGH, LOW
MICROSTEP4 = LOW, HIGH
MICROSTEP8 = HIGH, HIGH
MICROSTEP16 = HIGH, HIGH
```

```
MICROSTEP_MODES = {16, 16, 16, 16, 16}  # [1,2,4,8,16]

# Motor Current setting (Only functional when motor driver current
#  ref pins are connected to a digital trimpot on supported boards)
DIGIPOT_MOTOR_CURRENT = {135, 135, 135, 135, 135}  # Values 0-255 (RAMBO 135 = ~0.75A, 185 = ~1A)

# Number of channels available for I2C digipot, For Azteeg X3 Pro we have 8
DIGIPOT_I2C_NUM_CHANNELS = 8
# actual motor currents in Amps, need as many here as DIGIPOT_I2C_NUM_CHANNELS
DIGIPOT_I2C_MOTOR_CURRENTS = {1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0}

# ADDITIONAL FEATURES

CHDK_DELAY = 50  # How long in ms the pin should stay HIGH before going LOW again

SD_FINISHED_STEPPERRELEASE = True  # if sd support and the file is finished: disable steppers?
SD_FINISHED_RELEASECOMMAND = "M84 X Y Z E"  # You might want to keep the z enabled so your bed stays in place.

SDCARD_RATHERRECENTFIRST = 1  # reverse file order of sd card menu display.
#  Its sorted practically after the file system block order.

MM_PER_ARC_SEGMENT = 1
N_ARC_CORRECTION = 25

dropsegments = 5  # everything with less than this number of steps will
# be ignored as move and joined with the next movement

PS_ON_AWAKE = LOW
PS_ON_ASLEEP = HIGH
```

En el mòdul configurationStore es detallen les configuracions inicials de impressió con la configuració del PID, i les configuracions per defecte que no s'han modificat en el gcode:

```
def Config_ResetDefault():
    # // steps per sq second needto be updated to agreewith the units per sq second
    raspra.planner.reset_acceleration_rates()

    acceleration = raspra.configuration.DEFAULT_ACCELERATION
    retract_acceleration = raspra.configuration.DEFAULT_RETRACT_ACCELERATION
    minimumfeedrate = raspra.configuration_adv.DEFAULT_MINIMUMFEEDRATE
    minsegmenttime = raspra.configuration_adv.DEFAULT_MINSEGMENTTIME
    mintravelfeedrate = raspra.configuration_adv.DEFAULT_MINTRAVELFEEDRATE
    max_xy_jerk = raspra.configuration.DEFAULT_XYJERK
    max_z_jerk = raspra.configuration.DEFAULT_ZJERK
    max_e_jerk = raspra.configuration.DEFAULT_EJERK
    add_homing = [0, 0, 0]

    Kp = raspra.configuration.DEFAULT_Kp
    Ki = raspra.temperature.scalePID_i(raspra.configuration.DEFAULT_Ki)
    Kd = raspra.temperature.scalePID_d(raspra.configuration.DEFAULT_Kd)
    # // call updatePID(similar to when we have processed M301)
    raspra.temperature.updatePID()
    Kc = raspra.configuration_adv.DEFAULT_Kc

    volumetric_enabled = False
    filament_size = raspra.configuration.DEFAULT_NOMINAL_FILAMENT_DIA

    raspra.Marlin_main.calculate_volumetric_multipliers()

    raspra.Marlin_h.SERIAL_ECHOLNPGM("Hardcoded Default Settings Loaded")
    return (acceleration, retract_acceleration, minimumfeedrate, minsegmenttime, mintravelfeedrate, max_e_jerk,
            max_xy_jerk, max_z_jerk, add_homing, Kp, Ki, Kd, Kc, volumetric_enabled, filament_size)


Config_ResetDefault()
```

```
def Config_PrintSettings():
    axis_steps_per_unit = raspra.configuration.DEFAULT_AXIS_STEPS_PER_UNIT
    max_feedrate = raspra.configuration.DEFAULT_MAX_FEEDRATE
    max_acceleration_units_per_sq_second = raspra.configuration.DEFAULT_ACCELERATION
    acceleration = raspra.configuration.DEFAULT_ACCELERATION
    retract_acceleration = raspra.configuration.DEFAULT_RETRACT_ACCELERATION
    minimumfeedrate = raspra.configuration_adv.DEFAULT_MINIMUMFEEDRATE
    minsegmenttime = raspra.configuration_adv.DEFAULT_MINSEGMENTTIME
    mintravelfeedrate = raspra.configuration_adv.DEFAULT_MINTRAVELFEEDRATE
    max_xy_jerk = raspra.configuration.DEFAULT_XYJERK
    max_z_jerk = raspra.configuration.DEFAULT_ZJERK
    max_e_jerk = raspra.configuration.DEFAULT_EJERK

    # { // Always have this function, even with EEPROM_SETTINGS disabled, the current values will be shown
    raspra.Marlin_h.SERIAL_ECHOLNPGM("Steps per unit:")
    raspra.Marlin_h.SERIAL_ECHOPAIR("  M92 X", axis_steps_per_unit[0])
    raspra.Marlin_h.SERIAL_ECHOPAIR(" Y", axis_steps_per_unit[1])
    raspra.Marlin_h.SERIAL_ECHOPAIR(" Z", axis_steps_per_unit[2])
    raspra.Marlin_h.SERIAL_ECHOPAIR(" E", axis_steps_per_unit[3])
    raspra.Marlin_h.SERIAL_ECHOLN("")

    raspra.Marlin_h.SERIAL_ECHOLNPGM("Maximum feedrates (mm/s):")
    raspra.Marlin_h.SERIAL_ECHOPAIR("  M203 X", max_feedrate[0])
    raspra.Marlin_h.SERIAL_ECHOPAIR(" Y", max_feedrate[1])
    raspra.Marlin_h.SERIAL_ECHOPAIR(" Z", max_feedrate[2])
    raspra.Marlin_h.SERIAL_ECHOPAIR(" E", max_feedrate[3])
    raspra.Marlin_h.SERIAL_ECHOLN("")

    raspra.Marlin_h.SERIAL_ECHOLNPGM("Maximum Acceleration (mm/s2):")
    raspra.Marlin_h.SERIAL_ECHOPAIR("  M201 X", max_acceleration_units_per_sq_second[0])
    raspra.Marlin_h.SERIAL_ECHOPAIR(" Y", max_acceleration_units_per_sq_second[1])
    raspra.Marlin_h.SERIAL_ECHOPAIR(" Z", max_acceleration_units_per_sq_second[2])
    raspra.Marlin_h.SERIAL_ECHOPAIR(" E", max_acceleration_units_per_sq_second[3])
```

En el mòdul Language podem trobar la traducció a anglès dels missatges que es mostraran en el OctoPrint per mostrar dades de la impressió o errors que es puguin trobar:

```
MSG_POWERUP = "PowerUp"
MSG_EXTERNAL_RESET = " External Reset"
MSG_BROWNOUT_RESET = " Brown out Reset"
MSG_WATCHDOG_RESET = " Watchdog Reset"
MSG_SOFTWARE_RESET = " Software Reset"
MSG_AUTHOR = " | Author: "
MSG_CONFIGURATION_VER = " Last Updated: "
MSG_FREE_MEMORY = " Free Memory: "
MSG_PLANNER_BUFFER_BYTES = "  PlannerBufferBytes: "
MSG_OK = "ok"
MSG_WAIT = "wait"
MSG_STATS = "Stats: "
MSG_FILE_SAVED = "Done saving file."
MSG_ERR_LINE_NO = "Line Number is not Last Line Number+1, Last Line: "
MSG_ERR_CHECKSUM_MISMATCH = "checksum mismatch, Last Line: "
MSG_ERR_NO_CHECKSUM = "No Checksum with line number, Last Line: "
MSG_ERR_NO_LINENUMBER_WITH_CHECKSUM = "No Line Number with checksum, Last Line: "
MSG_FILE_PRINTED = "Done printing file"
MSG_BEGIN_FILE_LIST = "Begin file list"
MSG_END_FILE_LIST = "End file list"
MSG_INVALID_EXTRUDER = "Invalid extruder"
MSG_M104_INVALID_EXTRUDER = "M104 Invalid extruder "
MSG_M105_INVALID_EXTRUDER = "M105 Invalid extruder "
MSG_M200_INVALID_EXTRUDER = "M200 Invalid extruder "
MSG_M218_INVALID_EXTRUDER = "M218 Invalid extruder "
```

```
MSG_X_MIN = "x_min: "
MSG_X_MAX = "x_max: "
MSG_Y_MIN = "y_min: "
MSG_Y_MAX = "y_max: "
MSG_Z_MIN = "z_min: "
MSG_Z_MAX = "z_max: "
MSG_Z2_MIN = "z2_min: "
MSG_Z2_MAX = "z2_max: "
MSG_Z_PROBE = "z_probe: "
MSG_FILAMENT_RUNOUT_SENSOR = "filament: "
MSG_ERR_MATERIAL_INDEX = "M145 S<index> out of range (0-1)"
MSG_ERR_M355_NONE = "No case light"
MSG_ERR_M421_PARAMETERS = "M421 required parameters missing"
MSG_ERR_MESH_XY = "Mesh point cannot be resolved"
MSG_ERR_ARC_ARGS = "G2/G3 bad parameters"
MSG_ERR_PROTECTED_PIN = "Protected Pin"
MSG_ERR_M420_FAILED = "Failed to enable Bed Leveling"
MSG_ERR_M428_TOO_FAR = "Too far from reference point"
MSG_ERR_M303_DISABLED = "PIDTEMP disabled"
MSG_M119_REPORT = "Reporting endstop status"
MSG_ENDSTOP_HIT = "TRIGGERED"
MSG_ENDSTOP_OPEN = "open"
MSG_HOTEND_OFFSET = "Hotend offsets:"
MSG_DUPLICATION_MODE = "Duplication mode: "
MSG_SOFT_ENDSTOPS = "Soft endstops: "
```

En el mòdul pins_raspberry, estan relacionats els noms que rebran els pins GPIO que utilitzarem de la raspberry pi:

```
import RPi.GPIO as GPIO

GPIO.setmode(GPIO.BOARD)

KNOWN_BOARD = 1

LARGE_FLASH = True

X_STEP_PIN = 3
X_DIR_PIN = 5
X_ENABLE_PIN = 36
X_MIN_PIN = 31

Y_STEP_PIN = 11
Y_DIR_PIN = 13
Y_ENABLE_PIN = 7
Y_MIN_PIN = 16

Z_STEP_PIN = 32
Z_DIR_PIN = 29
Z_ENABLE_PIN = 15
Z_MIN_PIN = 18

E0_STEP_PIN = 24
E0_DIR_PIN = 26
E0_ENABLE_PIN = 22

FAN_PIN = 23

HEATER_0_PIN = 33  # EXTRUDER 1

TEMP_0_PIN = 19  # // ANALOG NUMBERING
TEMP_1_PIN = 21  # // ANALOG NUMBERING
```

En el mòdul Thermistorables es realitzen els càlculs necessaris per configurar els sensors de temperatura:

```
PtA = 3.9083E-3
PtB = -5.775E-7


def PtRt(T, R0):
    (R0 * (1.0 + PtA * T + PtB * T * T))


def PtAdVal(T, R0, Rup):
    a = (1024 / (Rup / PtRt(T, R0) + 1))
    return a


def PtLine(T, R0, Rup):
    (PtAdVal(T, R0, Rup) * OVERSAMPLENR, T)


_TT_NAME = temptable_1
TT_NAME = _TT_NAME

if raspra.configuration_adv.THERMISTORHEATER_0:
    HEATER_0_TEMPTABLE = raspra.configuration_adv.THERMISTORHEATER_0
    HEATER_0_TEMPTABLE_LEN = (len(HEATER_0_TEMPTABLE) / len(*HEATER_0_TEMPTABLE))
else:
    if raspra.configuration_adv.HEATER_0_USES_THERMISTOR:
        print('error No heater 0 thermistor table specified')
    else:  # HEATER_0_USES_THERMISTOR
        HEATER_0_TEMPTABLE = None
        HEATER_0_TEMPTABLE_LEN = 0

# Set the high and low raw values for the heater, this indicates which raw value is a high or low temperature
if raspra.configuration_adv.HEATER_0_USES_THERMISTOR:
```

```
    # In case of a thermistor the highest temperature results in the lowest ADC value
    HEATER_0_RAW_HI_TEMP = 0
    HEATER_0_RAW_LO_TEMP = 16383
else:
    HEATER_0_RAW_HI_TEMP = 16383
    HEATER_0_RAW_LO_TEMP = 0

# Set the high and low raw values for the heater, this indicates which raw value is a high or low temperature
#                     #In case of an thermocouple the highest temperature results in the highest ADC value
HEATER_1_RAW_HI_TEMP = 16383
HEATER_1_RAW_LO_TEMP = 0

if raspra.configuration_adv.THERMISTORBED:
    BEDTEMPTABLE = raspra.configuration_adv.THERMISTORBED
    BEDTEMPTABLE_LEN = (len(BEDTEMPTABLE) / len(*BEDTEMPTABLE))
else:
    if raspra.configuration_adv.BED_USES_THERMISTOR:
        print('error No bed thermistor table specified')

# Set the high and low raw values for the heater, this indicates which raw value is a high or low temperature
if raspra.configuration_adv.BED_USES_THERMISTOR:
    # In case of a thermistor the highest temperature results in the lowest ADC value
    HEATER_BED_RAW_HI_TEMP = 0
    HEATER_BED_RAW_LO_TEMP = 16383
else:  # In case of an thermocouple the highest temperature results in the highest ADC value
    HEATER_BED_RAW_HI_TEMP = 16383
    HEATER_BED_RAW_LO_TEMP = 0
```

En el mòdul steppers es realitza el codi per generar els impulsos elèctrics específics per a realitzar el moviment que demana Marlin_main adaptats en el mòdul planner i segons la configuració dels nostres controladors de motors.

```
endstops_trigsteps = [0, 0, 0]
endstop_x_hit = False
endstop_y_hit = False
endstop_z_hit = False

old_x_min_endstop = False
old_x_max_endstop = False
old_y_min_endstop = False
old_y_max_endstop = False
old_z_min_endstop = False
old_z_max_endstop = False

check_endstops = True

count_position = [0, 0, 0, 0]
count_direction = [1, 1, 1, 1]


# //===========================================================================
# //=============================functions         ============================
# //===========================================================================

def checkHitEndstops(endstop_x_hit, endstop_y_hit, endstop_z_hit):
    if endstop_x_hit or endstop_y_hit or endstop_z_hit:
        ser.open()
        ser.write(raspra.language.MSG_ENDSTOPS_HIT)
```

```
def endstops_hit_on_purpose():
    endstop_x_hit = False
    endstop_y_hit = False
    endstop_z_hit = False
    return endstop_x_hit,  endstop_y_hit, endstop_z_hit

def enable_endstops(check):
  check_endstops = check
  return check_endstops

def st_wake_up():
  #  TCNT1 = 0;
  return

def step_wait():
    a=(6,5,4,3,2,1)
    for i in a :
        return

def calc_timer(step_rate) :
    timer=0
    if(step_rate > raspra.configuration_adv.MAX_STEP_FREQUENCY) :
        step_rate = raspra.configuration_adv.MAX_STEP_FREQUENCY

    if(step_rate > 20000) :
        # If steprate > 20kHz >> step 4 times
        step_rate = (step_rate >> 2)&0x3fff
        step_loops = 4
```

```
def st_init():

    digipot_init() # Initialize Digipot Motor Current
    microstep_init() # Initialize MicrosteppingPins

#Initialize Dir Pins
    if raspra.pins_raspberry.X_DIR_PIN > -1:
        GPIO.setup(raspra.pins_raspberry.X_DIR_PIN, GPIO.OUT)

    if raspra.pins_raspberry.Y_DIR_PIN > -1:
        GPIO.output(raspra.pins_raspberry.Y_DIR_PIN)

    if raspra.pins_raspberry.Z_DIR_PIN > -1:
        GPIO.output(raspra.pins_raspberry.Z_DIR_PIN)

    if raspra.pins_raspberry.E0_DIR_PIN > -1:
        GPIO.output(raspra.pins_raspberry.E0_DIR_PIN)

#Initialize Enable Pins - steppers default to disabled.

    if raspra.pins_raspberry.X_ENABLE_PIN > -1:
        GPIO.output(raspra.pins_raspberry.X_ENABLE_PIN)
        if not raspra.configuration.X_ENABLE_ON :
            GPIO.output(raspra.pins_raspberry.X_ENABLE_PIN, GPIO.HIGH)
    if raspra.pins_raspberry.Y_ENABLE_PIN > -1:
        GPIO.output(raspra.pins_raspberry.Y_ENABLE_PIN)
        if not raspra.configuration.Y_ENABLE_ON :
            GPIO.output(raspra.pins_raspberry.Y_ENABLE_PIN, GPIO.HIGH)
```

```
if raspra.pins_raspberry.X_MIN_PIN > -1:
    GPIO.input(raspra.pins_raspberry.X_MIN_PIN)
    if raspra.configuration.ENDSTOPPULLUP_XMIN:
        GPIO.output(raspra.pins_raspberry.X_MIN_PIN, GPIO.HIGH)

if raspra.pins_raspberry.Y_MIN_PIN > -1:
    GPIO.input(raspra.pins_raspberry.Y_MIN_PIN)
    if raspra.configuration.ENDSTOPPULLUP_YMIN:
        GPIO.output(raspra.pins_raspberry.Y_MIN_PIN, GPIO.HIGH)

if raspra.pins_raspberry.Z_MIN_PIN > -1:
    GPIO.input(raspra.pins_raspberry.Z_MIN_PIN)
    if raspra.configuration.ENDSTOPPULLUP_ZMIN:
        GPIO.output(raspra.pins_raspberry.Z_MIN_PIN, GPIO.HIGH)

# Initialize Step Pins

if raspra.pins_raspberry.X_STEP_PIN > -1:
    GPIO.output(raspra.pins_raspberry.X_STEP_PIN)
    GPIO.output(raspra.pins_raspberry.X_STEP_PIN )
    raspra.Marlin_h.disable_x()

if raspra.pins_raspberry.Y_STEP_PIN > -1:
    GPIO.output(raspra.pins_raspberry.Y_STEP_PIN)
    GPIO.output(raspra.pins_raspberry.Y_STEP_PIN )
    raspra.Marlin_h.disable_y()

if raspra.pins_raspberry.Z_STEP_PIN > -1:
    GPIO.output(raspra.pins_raspberry.Z_STEP_PIN)
    GPIO.output(raspra.pins_raspberry.Z_STEP_PIN )
    raspra.Marlin_h.disable_z()
if raspra.pins_raspberry.E0_STEP_PIN > -1:
```

```
def microstep_ms(driver, ms1, ms2):
  if(ms1 > -1):
      def case0 ():
          X_MS1_PIN=ms1
          return
      def case1():
          Y_MS1_PIN= ms1
          return
      def case2():
          Z_MS1_PIN= ms1
          return
      def case3():
          E0_MS1_PIN= ms1
          return

      try:
          ({'0':case0,'1':case1,'2':case2,'3':case3}[driver])()
      finally:
          print ('Error')

      if raspra.pins_raspberry.E1_MS1_PIN > -1:
           E1_MS1_PIN=ms1

  if(ms2 > -1):
      def case0 ():
          X_MS2_PIN= ms1
          return
      def case1():
          Y_MS2_PIN= ms1
          return
      def case2():
          Z_MS2_PIN= ms1
```

```
def microstep_readings():
      raspra.Marlin_h.SERIAL_PROTOCOLPGM("MS1,MS2 Pins\n")
      raspra.Marlin_h.SERIAL_PROTOCOLPGM("X: ")
      raspra.Marlin_h.SERIAL_PROTOCOL(   GPIO.input(raspra.pins_raspberry.X_MS1_PIN))
      raspra.Marlin_h.SERIAL_PROTOCOLLN( GPIO.input(raspra.pins_raspberry.X_MS2_PIN))
      raspra.Marlin_h.SERIAL_PROTOCOLPGM("Y: ")
      raspra.Marlin_h.SERIAL_PROTOCOL(   GPIO.input(raspra.pins_raspberry.Y_MS1_PIN))
      raspra.Marlin_h.SERIAL_PROTOCOLLN( GPIO.input(raspra.pins_raspberry.Y_MS2_PIN))
      raspra.Marlin_h.SERIAL_PROTOCOLPGM("Z: ")
      raspra.Marlin_h.SERIAL_PROTOCOL(   GPIO.input(raspra.pins_raspberry.Z_MS1_PIN))
      raspra.Marlin_h.SERIAL_PROTOCOLLN( GPIO.input(raspra.pins_raspberry.Z_MS2_PIN))
      raspra.Marlin_h.SERIAL_PROTOCOLPGM("E0: ")
      raspra.Marlin_h.SERIAL_PROTOCOL(   GPIO.input(raspra.pins_raspberry.E0_MS1_PIN))
      raspra.Marlin_h.SERIAL_PROTOCOLLN( GPIO.input(raspra.pins_raspberry.E0_MS2_PIN))
      if raspra.pins_raspberry.E1_MS1_PIN > -1:
          raspra.Marlin_h.SERIAL_PROTOCOLPGM("E1: ")
          raspra.Marlin_h.SERIAL_PROTOCOL(   GPIO.input(raspra.pins_raspberry.E1_MS1_PIN))
          raspra.Marlin_h.SERIAL_PROTOCOLLN( GPIO.input(raspra.pins_raspberry.E1_MS2_PIN))

def WRITE_E_STEP(v):
    GPIO.output(raspra.pins_raspberry.E0_STEP_PIN, v)
def NORM_E_DIR():
    GPIO.output(raspra.pins_raspberry.E0_DIR_PIN, not raspra.configuration.INVERT_E0_DIR)
def REV_E_DIR():
    GPIO.output(raspra.pins_raspberry.E0_DIR_PIN, raspra.configuration.INVERT_E0_DIR)
```

En el mòdul motion_control es converteixen els moviments demanats per Marlin_main en coordenades correctes a partir de càlculs.

```
import raspra.stepper
import raspra.Marlin_h
import raspra.planner
import raspra.configuration_adv
import math
import raspra.Marlin_main


# // The arc is approximated iby igenerating ia huge inumber of tiny, linear segments.The length of each
# // segment is configured in settings.mm_per_arc_segment.
def mc_arc(position, target, offset, axis_0, axis_1, axis_linear, feed_rate, radius, isclockwise, extruder):
    # // int acceleration_manager_was_enabled = plan_is_acceleration_manager_enabled();
    # // plan_set_acceleration_manager_enabled(false); // disable acceleration management for the duration of the arc
    center_axis0 = position[0] + offset[0]
    center_axis1 = position[1] + offset[1]
    linear_travel = target[axis_linear] - position[axis_linear]
    extruder_travel = target[3] - position[3]
    r_axis0 = -offset[0]  # Radius vector from center to current location
    r_axis1 = -offset[1]
    rt_axis0 = target[0] - center_axis0
    rt_axis1 = target[1] - center_axis1

    # // CCW angle between position and target from circle center.Only one atan2() trig computation required.

    angular_travel = math.atan2(r_axis0 * rt_axis1 - r_axis1 * rt_axis0, r_axis0 * rt_axis0 + r_axis1 * rt_axis1)
    if angular_travel < 0:
        angular_travel += 2 * math.pi
    if isclockwise:
        angular_travel -= 2 * math.pi
```

```
# // 20141002:full circle for G03 did not work, e.g.G03 X80 Y80 I20 J0 F2000
# is giving an Angle of zero so head is not moving
# // to compensate when start pos = target pos & & angle is zero -> angle = 2Pi
if position[axis_0] == target[axis_0] and position[axis_1] == target[axis_1] and angular_travel == 0:
    angular_travel += 2 * math.pi
# // end fix G03

millimeters_of_travel = math.hypot(angular_travel * radius, math.fabs(linear_travel))
if millimeters_of_travel < 0.001:
    return
segments = math.floor(millimeters_of_travel / raspra.configuration_adv.MM_PER_ARC_SEGMENT)
if segments == 0:
    segments = 1

# / *
# // Multiply inverse feed_rate to compensate for the fact that this movement is approximated
# // by a number of discrete segments.The inverse feed_rate should be correct for the sum of
# // all segments.
#     if (invert_feed_rate):
#         feed_rate *= segments

theta_per_segment = angular_travel / segments
linear_per_segment = linear_travel / segments
extruder_per_segment = extruder_travel / segments

# // Vector rotation matrix values
cos_T = 1 - 0.5 * theta_per_segment * theta_per_segment  # Small angle approximation
sin_T = theta_per_segment

count = 0
```

```
# // Initialize the extruder axis
arc_target = position[3]

for i in range(0, segments):

    if count < raspra.configuration_adv.N_ARC_CORRECTION:
        r_axisi = r_axis0 * sin_T + r_axis1 * cos_T
        r_axis0 = r_axis0 * cos_T - r_axis1 * sin_T
        r_axis1 = r_axisi
        count = count + 1
    else:

        cos_Ti = math.cos(i * theta_per_segment)
        sin_Ti = math.sin(i * theta_per_segment)
        r_axis0 = -offset[axis_0] * cos_Ti + offset[axis_1] * sin_Ti
        r_axis1 = -offset[axis_0] * sin_Ti - offset[axis_1] * cos_Ti
        count = 0

    arc_target[0] = center_axis0 + r_axis0
    arc_target[1] = center_axis1 + r_axis1
    arc_target[2] += linear_per_segment
    arc_target[3] += extruder_per_segment

    raspra.Marlin_main.clamp_to_software_endstops(arc_target)
    raspra.planner.plan_buffer_line(arc_target[0], arc_target[1], arc_target[2], arc_target[3], feed_rate, extruder)

raspra.planner.plan_buffer_line(target[0], target[1], target[2], target[3], feed_rate, extruder)
```

En el mòdul planner es detallen els moviments dels quatre motors convertint les distancies i les acceleracions en els impulsos elèctrics necessaris per el correcte moviment dels motors seguint les seves especificacions tècniques, ja que han sigut detallats les característiques en el mòduls steppers.

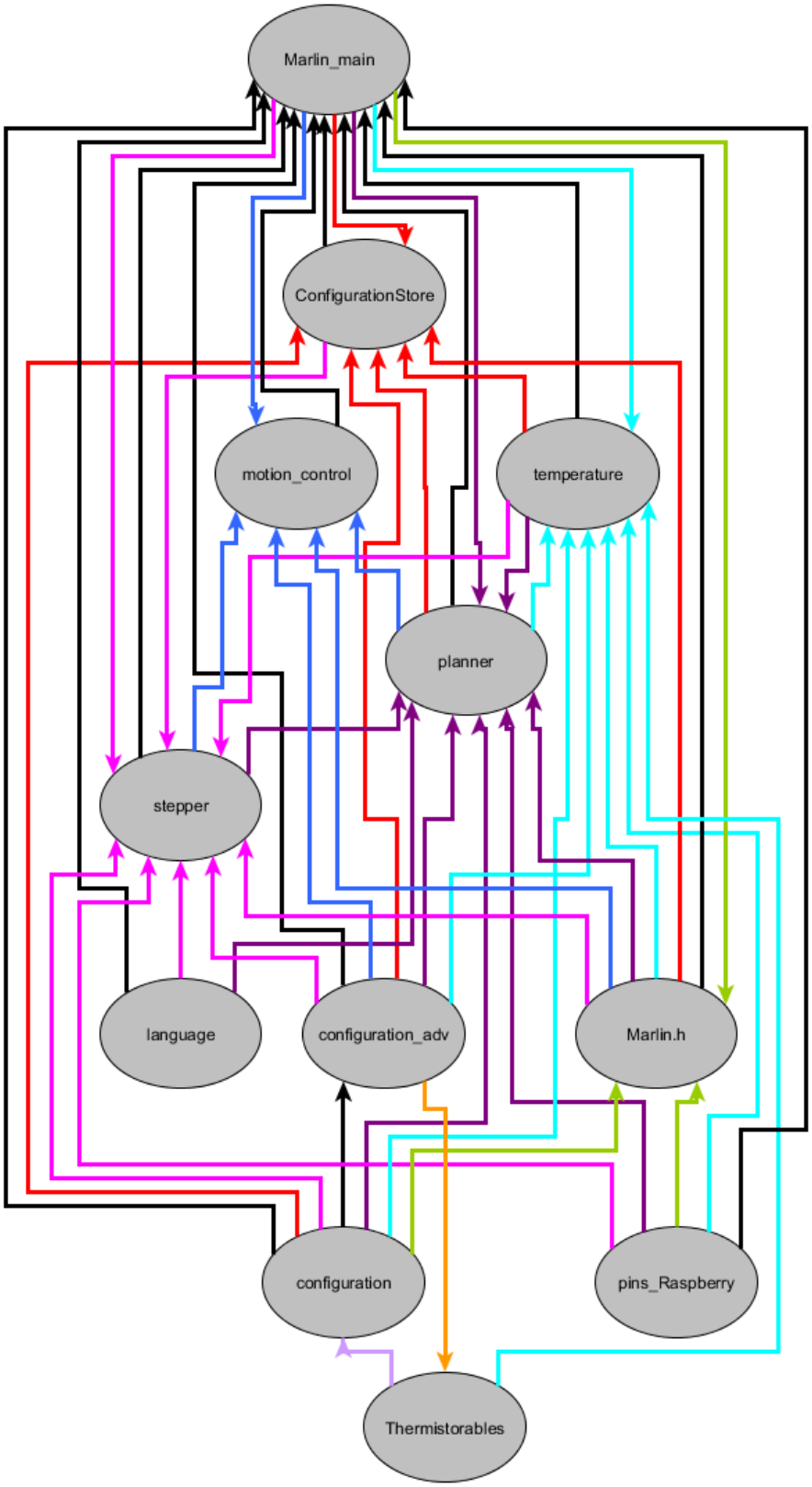
```
autotemp_max = 250
autotemp_min = 210
autotemp_factor = 0.1
autotemp_enabled = False

g_uc_extruder_last_move = [0, 0, 0]

block_buffer = [raspra.configuration_adv.BLOCK_BUFFER_SIZE]

extrude_min_temp = raspra.configuration.EXTRUDE_MINTEMP


def next_block_index(block_index):
    block_index = block_index + 1
    if block_index == raspra.configuration_adv.BLOCK_BUFFER_SIZE:
        block_index = 0
    return block_index


def prev_block_index(block_index):
    if block_index == 0:
        block_index = raspra.configuration_adv.BLOCK_BUFFER_SIZE
    block_index = block_index - 1
    return block_index
```

```
def intersection_distance(initial_rate, final_rate, acceleration, distance):
    if acceleration != 0:
        return (2.0 * acceleration * distance - initial_rate * initial_rate + final_rate * final_rate) \
               / (4.0 * acceleration)
    else:
        return 0.0  # acceleration was 0, set intersection distance to 0


def calculate_trapezoid_for_block(block, entry_factor, exit_factor):
    initial_rate = math.ceil(nominal_rate * entry_factor)  # (step/min)
    final_rate = math.ceil(nominal_rate * exit_factor)  # (step/min)

    # // Limit minimal step rate (Otherwise the timer will overflow.)
    if (initial_rate < 120):
        initial_rate = 120
    if (final_rate < 120):
        final_rate = 120

    acceleration = acceleration_st
    accelerate_steps = math.ceil(estimate_acceleration_distance(initial_rate, nominal_rate, acceleration))
    decelerate_steps = math.floor(estimate_acceleration_distance(nominal_rate, final_rate, -acceleration))

    # // Calculate the size of Plateau of Nominal Rate.
    plateau_steps = step_event_count - accelerate_steps - decelerate_steps
```

```
# // Calculate the size of Plateau of Nominal Rate.
plateau_steps = step_event_count - accelerate_steps - decelerate_steps

# // Is the Plateau of Nominal Rate smaller than nothing? That means no cruising, and we will
# // have to use intersection_distance() to calculate when to abort acceleration and start braking
# // in order to reach the final_rate exactly at the end of this block.
if plateau_steps < 0:
    accelerate_steps = math.ceil(intersection_distance(initial_rate, final_rate, acceleration,
                                                       step_event_count))
    accelerate_steps = max(accelerate_steps, 0)  # // Check limits due to numerical round-off
    accelerate_steps = min(accelerate_steps,
                           step_event_count)  # (We can cast here to unsigned, because the above
    #  line ensures that we are above zero)
    plateau_steps = 0

if busy == False:  # { // Don't update variables if block is busy.
    accelerate_until = accelerate_steps
    decelerate_after = accelerate_steps + plateau_steps
    initial_rate = initial_rate
    final_rate = final_rate

    def max_allowable_speed(acceleration, target_velocity, distance):
        return math.sqrt(target_velocity * target_velocity - 2 * acceleration * distance)
```

```
# // Enable extruder(s)
if (steps_e != 0):

    if (raspra.configuration.DISABLE_INACTIVE_EXTRUDER):  # // enable only selected extruder


        if (g_uc_extruder_last_move[0] > 0):
            g_uc_extruder_last_move[0] = g_uc_extruder_last_move[0] - 1
        if (g_uc_extruder_last_move[1] > 0):
            g_uc_extruder_last_move[1] = g_uc_extruder_last_move[1] - 1
        if (g_uc_extruder_last_move[2] > 0):
            g_uc_extruder_last_move[2] = g_uc_extruder_last_move[2] - 1


        def case0():
            raspra.Marlin_h.enable_e0()
            g_uc_extruder_last_move[0] = raspra.configuration_adv.BLOCK_BUFFER_SIZE * 2
            return


        def case1():
            g_uc_extruder_last_move[1] = raspra.configuration_adv.BLOCK_BUFFER_SIZE * 2

            if (g_uc_extruder_last_move[0] == 0):
                raspra.Marlin_h.disable_e0()
            return


        def case2():
            g_uc_extruder_last_move[2] = raspra.configuration_adv.BLOCK_BUFFER_SIZE * 2
```

Tots els mòduls son necessaris per el correcte funcionament de la impressora, ja que tots estan relacionats per les variables que es declaren en cadascun del mòduls i per les funcions que es defineixen. Seguidament es podrà observar en la Imatge 39: Ordinograma del codi per la raspberry pi.



Imatge 39: Ordinograma del codi per la raspberry pi

Amb tots els mòduls codificats i detallats s'ha procedit a instal·lar el Octoprint.

5.4. Octoprint

Octoprint es un software amb el qual es pot mantenir una connexió amb l'estat de la impressora via internet, per tant ens podrem comunicar amb la impressora sempre que tinguem internet.

Aquest programa es gratuït, de software lliure i es pot descarregar directament des de la web.

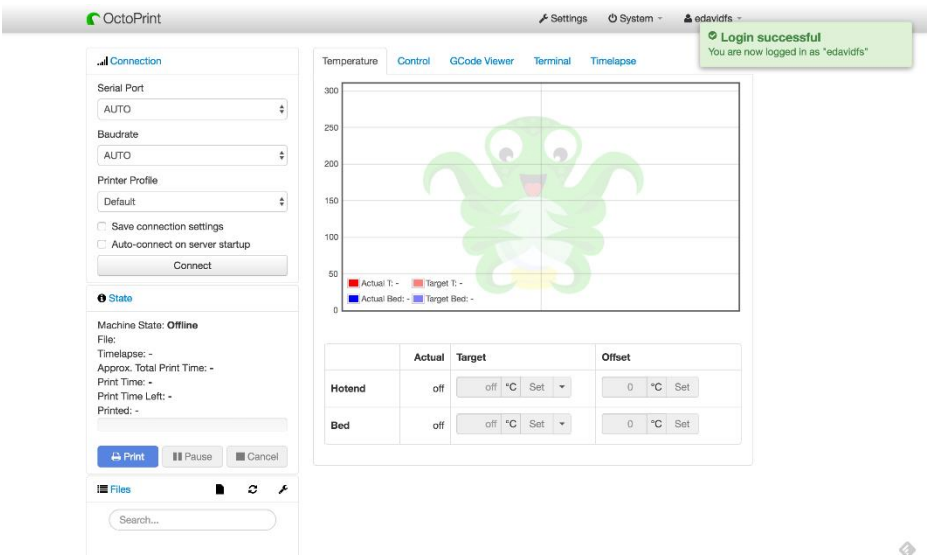
Un cop el tenim descarregat es procedirà a la instal·lació.

Per tal d'instal·lar-lo a la Raspberry Pi primerament s'haurà de transferir la imatge de OctoPrint a una sd, i es farà la configuració del wifi a partir de l'arxiu octopi-network.txt:

```
iface wlan0 inet manual
  wpa-ssid nombre-de-la-red-wifi
  wpa-psk paswword
```

Seguidament s'instal·larà el hardware, aquest procés es realitzarà connectant la sd a la Raspberry Pi junt a la Raspa i alimentada.

I un cop ja tenim això realitzat ens podem connectar des de un dispositiu remot amb connexió a internet a la direcció: http://octopi.local i creant un compte per la impressora.

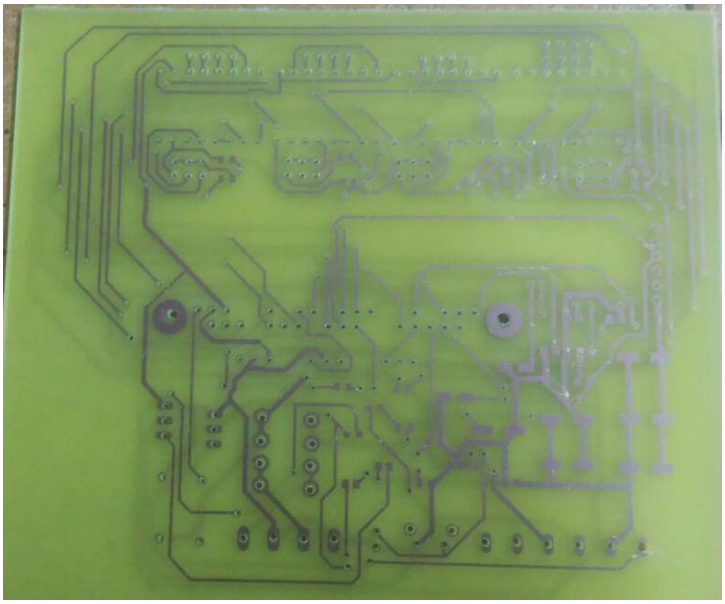


Imatge 40: Octoprint vist des de l'ordinador

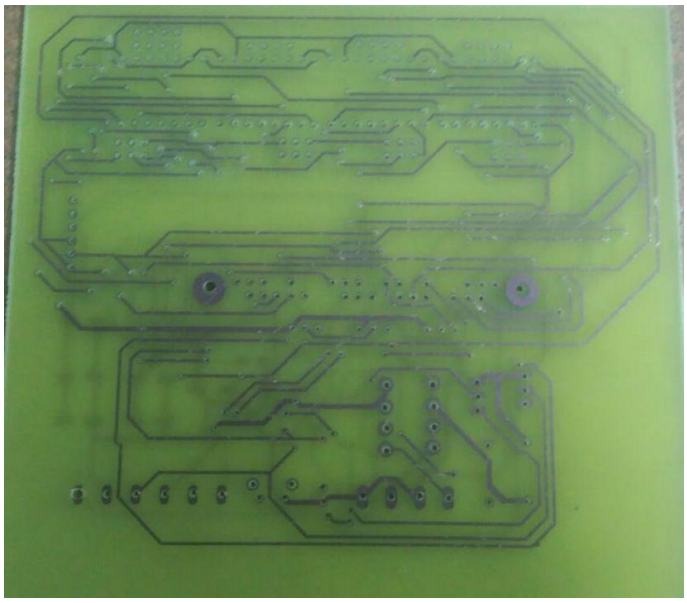
6. Probes i resultats

6.1. Muntatge

Per a realitzar el muntatge de la PCB primer s'ha procedit a realitzar tots els forats pertinents per poder encaixar els components que ho precisaven.

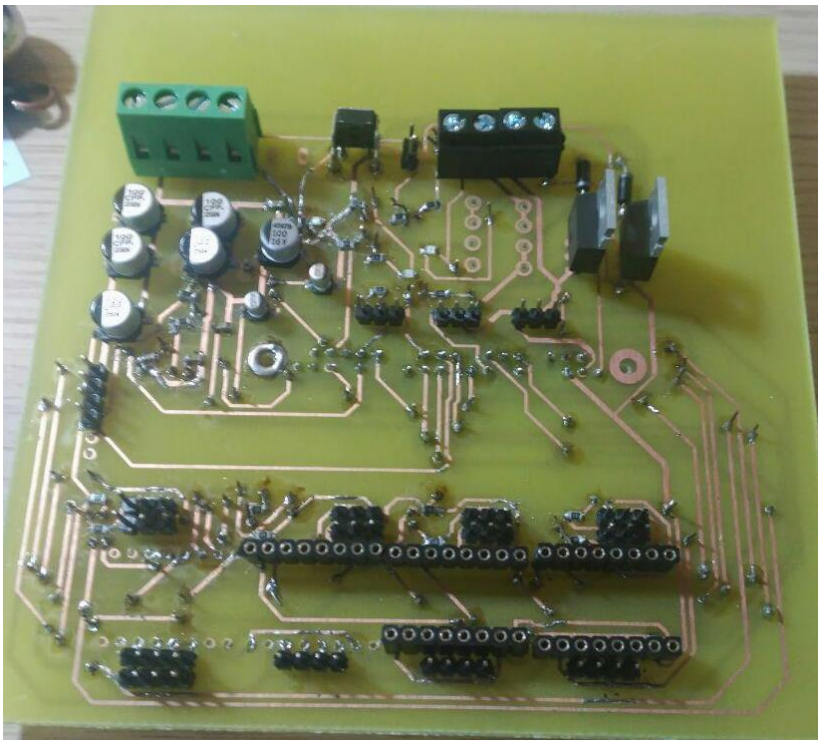


Imatge 41: Top PCB amb els forats realitzats

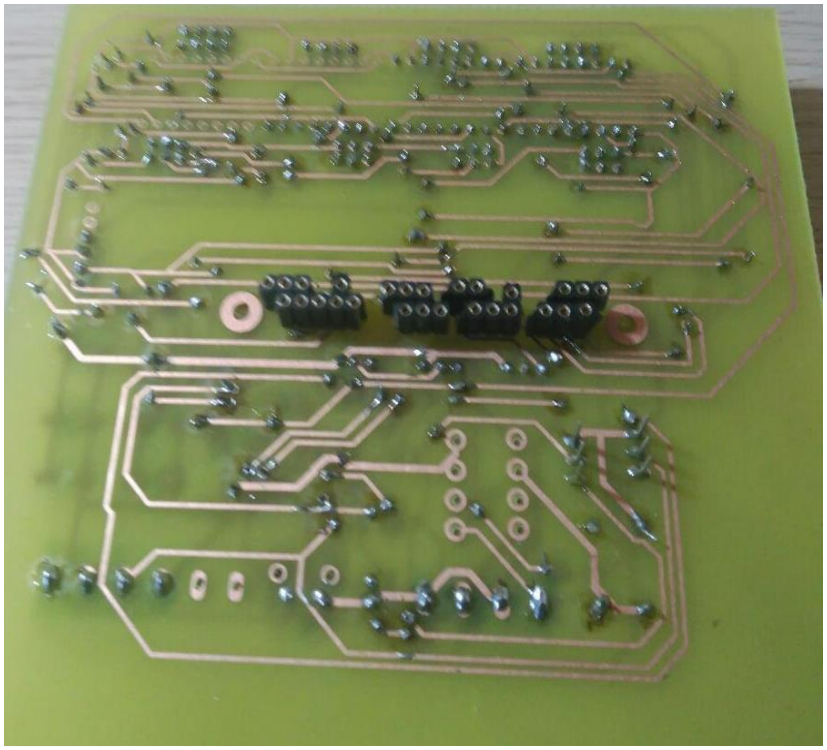


Imatge 42: Bottom PCB amb els forats realitzats

Seguidament es van soldar tots els components a la PCB.



Imatge 43: Top PCB amb els components soldats



Imatge 44: Bottom PCB amb els components soldats

Un cop es tenia la PCB llesta es va procedir a la instal·lació del software i l'Octoprint a la Raspberry Pi, ja que el programa s'havia realitzat des de l'ordinador.

6.2. Resultats

Finalment, es va procedir a comprovar que alimentant la raspberry pi des de la placa Raspra la Raspberry funcionava correctament.

Un cop el primer pas s'havia realitzat amb èxit, des de la raspberry, amb l'ajuda d'una pantalla auxiliar tàctil, es va procedir a carregar un arxiu al OctoPrint i seguidament, es va procedir a imprimir una peça de prova que ja s'havia realitzat primerament amb els components electrònics de sèrie, es dir, amb l'Arduino i la ramps.

La impressió es va aconseguir acabar amb èxit, per tant, el objectiu del projecte s'ha aconseguit.

Imatge 45: Resultat primera impressió

Imatge 46: Resultat primera impressió des de una altre perspectiva

Conclusions

En aquest punt s'extrauran les conclusions per haver finalitzat aquest projecte.

En primer punt es vol destacar la complexitat que s'ha trobat en els punts clau del projecte a mesura que s'anava avançant. Com en la adaptació del circuit a les característiques de la raspberry i dels equips subministrats per la universitat.

Però sens dubte la major dificultat ha residit en traduir el llenguatge Arduino en python, ja que s'han trobat una sèrie de característiques bàsiques que eren totalment diferents en python, o no existien a priori i s'ha hagut de trobar la manera de realitzar la mateixa funció.

També s'ha trobat dificultat per instal·lar mòduls externs al programa de python per poder-los utilitzar en el codi, com per exemple el mòdul RPi.GPIO.

Però finalment s'ha pogut superar les dificultats trobades i s'ha pogut dur a terme el projecte amb èxit.

De cara a futures millores es podria realitzar la unió del programa que converteix el arxiu en 3d a gcode directament amb el OctoPrint, així simplement amb el arxiu en 3d es podria imprimir directament. Aquesta feina es podria realitzar ja que la raspberry té capacitat suficient per realitzar aquesta tasca.

Pressupost i/o Anàlisi Econòmica

Per a la realització del estudi econòmic s'han tingut en compte els costos associats a la investigació i desenvolupament del projecte i el cost dels materials utilitzats.

El cost total del projecte s'ha desglossat en el cost del disseny conceptual y al cost del disseny de la PCB y del codi del programa.

Per el càlcul del cost de la PCB s'han tingut en compte els components que s'han hagut d'adquirir per a la seva realització, els quals estan redactats a la Taula 2: Cost total de la PCB:

Quantitat	Valor	Preu unitari (€)	Preu (€)
6	100 nF	0.2	1.2
2	10 uF	0.14	0.28
1	100 nF 0805	0.05	0.05
3	4.7 nF	0.05	0.15
2	1N4004	0.04	0.08
2	4.7KΩ	0.03	0.06
6	100 kΩ	0.03	0.18
1	220 Ω	0.03	0.03
2	10 Ω	0.03	0.06
4	10k Ω	0.03	0.12
2	1.8 KΩ	0.03	0.06
2	FUSE_MCCQ-122	5	10
3	CHIP-LED0805	0.2	0.6
2	TO220BV	0.55	1.1
1	B3F-31XX	0.12	0.12
4	Female Pin Strip *40	2	8
2	Male Pin Strip *40	0.5	1
1	PLACA POSITIVA DOBLE CARA 125X165	9.45	9.45
TOTAL			**19**

Taula 2: Cost total de la PCB

Per tant, a continuació es mostrarà la taula del cost total del projecte tenint en compte tots els costos esmenats anteriorment a la taula...:

Concepte	Quantitat	Cost	Import (€)
Cost d'enginyeria			
Recerca d'informació	60 h	20€/h	1200
Disseny PCB Raspra	170 h	30€/h	5100
Creació codi programa	320	30€/h	9600
Fabricació PCB	1	19	19
Construcció projecte	60	20€/h	1200
TOTAL			**17.119**

Taula 3: Cost total del projecte

Per a realitzar el pressupost no s'han tingut en compte els costos de transport.

Bibliografia

1. Downey, A., Elkner, J. i Meyers, C. *Aprenda a Pensar Como un Programador* [en línia]. Green Tea. Wellesley, Massachusetts: 2002. ISBN 0971677506. Disponible a: thinkpython.com

2. RepRapWiki. G-code. A: [en línia]. 2015. Disponible a: http://reprap.org/wiki/G-code

3. RepRapWiki. RAMPS 1.4. A: [en línia]. 2016. Disponible a: http://reprap.org/wiki/RAMPS_1.4

4. C. Escobar. *SLS y SLA_ que son y en qué se distinguen* [en línia]. 2013. Impresoras 3d,2013. Disponible a: https://impresoras3d.com/blogs/noticias/102843079-sls-y-sla-que-son-y-en-que-se-distinguen

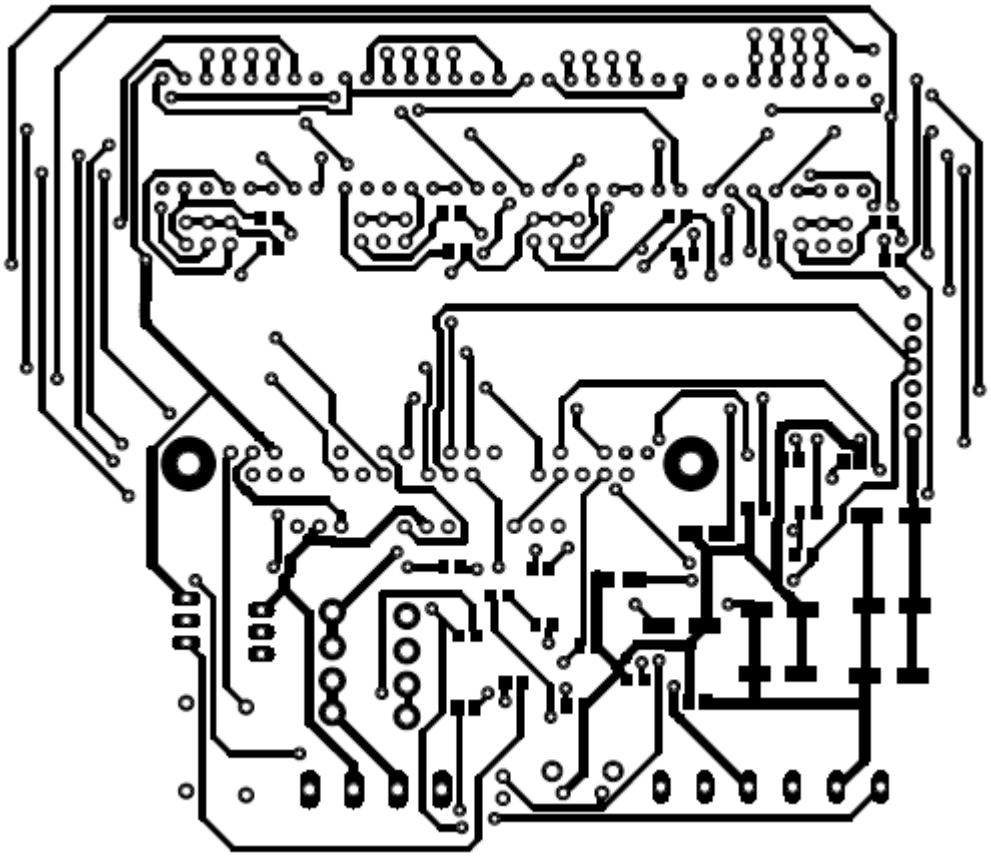
5. LibrosWeb. *Creando módulos empaquetados* [en línia]. 2016. 2016. Disponible a: http://librosweb.es/libro/python/capitulo_3/creando_modulos_empaquetados.html

Annex A: Components PCB

Nombre	Valor	Packaging
C2	100 nF	C0805
C3	100 uF	153CLV-0605
C4	100 uF	153CLV-0605
C5	10 uF	153CLV-0405
C6	100 uF	153CLV-0605
C7	100 uF	153CLV-0605
C8	10 uF	153CLV-0405
C9	100 uF	153CLV-0605
C10	100 uF	153CLV-0605
C11	4.7 nF	C0805
C13	4.7 nF	C0805
C15	4.7 nF	C0805
D1	1N4004	DO41-10
D2	1N4004	DO41-10
R1	4.7KΩ	R0805
R2	100 kΩ	R0805
R4	100 kΩ	R0805
R5	100 kΩ	R0805
R6	100 kΩ	R0805
R7	4,7 kΩ	R0805
R8	100 kΩ	R0805
R10	100 kΩ	R0805
R12	220 Ω	R0805
R14	10 Ω	M0805
R15	10 Ω	R0805
R16	10k Ω	R0805
R18	10 kΩ	R0805
R19	10 kΩ	R0805
R20	10 kΩ	R0805
R23	1.8 KΩ	R0805
R24	1.8 KΩ	R0805
F1		FUSE_MCCQ-122
F2		FUSE_MCCQ-122
LED1		CHIP-LED0805
LED2		CHIP-LED0805
LED3		CHIP-LED0805
Q2	STP55NF06L	TO220BV

Q3	STP55NF06L	TO220BV
S1	Interruptor	B3F-31XX
Z-, Y-, X-	3	Male Pin Strip
JP7	6	Male Pin Strip
JP2, JP4, JP5, JP6	2*3	Male Pin Strip
EO-MOT, X-MOT, Y-MOT, Z-MOT 1, Z-MOT	4	Male Pin Strip
RESET	2	Male Pin Strip
J8	27	Female Pin Strip
EO-DRIV, X-DRIV, Y-DRIV, Z-DRIV	8*2	Female Pin Strip
X4		MSTBA4
U$2		282837-6

Annex B: Plànols

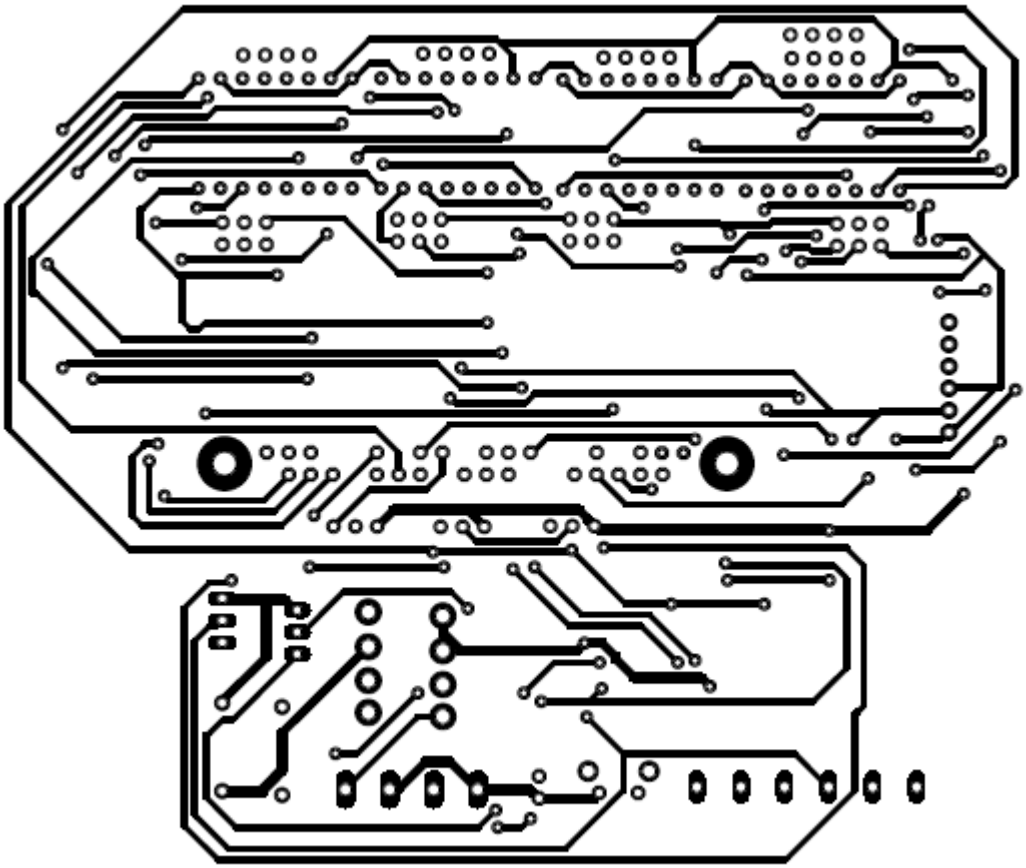


DISSENY ELECTRÒNIC I POSTA EN MARXA D'UNA IMPRESSORA 3D AMB RASPBERRY

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Escola d'Enginyeria de Barcelona Est

Títol: Top PCB raspra

	Nom	Data	Nº de dibuix:	Escala:
Dibuixat per:	Meritxell Rodríguez Torrejón	28-12-2017	1	1:1
Comprovat per:	Sebastian Tornil Sin	3-01-2018	Observacions:	



DISSENY ELECTRÒNIC I POSTA EN MARXA D'UNA IMPRESSORA 3D AMB RASPBERRY

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Escola d'Enginyeria de Barcelona Est

Títol: Bottom PCB raspra

	Nom	Data	Nº de dibuix:	Escala:
Dibuixat per:	Meritxell Rodríguez Torrejón	28-12-2017	2	1:1
Comprovat per:	Sebastian Tornil Sin	3-01-2018	Observacions:	